Theory and Applications of Categories, Vol. 6, No. 4, pp. 47–64.

A NOTE ON REWRITING THEORY FOR UNIQUENESS OF ITERATION

Dedicated to our friend and colleague Jim Lambek

M. OKADA[1] AND P. J. SCOTT[2]

Abstract. Uniqueness for higher type term constructors in lambda calculi (e.g. surjective pairing for product types, or uniqueness of iterators on the natural numbers) is easily expressed using universally quantified conditional equations. We use a technique of Lambek[18] involving Mal'cev operators to equationally express uniqueness of iteration (more generally, higher-order primitive recursion) in a simply typed lambda calculus, essentially Gödel's T [29, 13]. We prove the following facts about typed lambda calculus with uniqueness for primitive recursors: (i) It is undecidable, (ii) Church-Rosser fails, although ground Church-Rosser holds, (iii) strong normalization (termination) is still valid. This entails the undecidability of the coherence problem for cartesian closed categories with strong natural numbers objects, as well as providing a natural example of the following computational paradigm: a non-CR, ground CR, undecidable, terminating rewriting system.

1. Introduction

Consider the usual primitive recursion equations of the addition function: $x + 0 = x$, $x + (Sy) = S(x + y)$. How do we know these uniquely specify addition? More generally, consider a possibly higher-order primitive recursor, $\mathcal{R}ahxy$, where $a : A \Rightarrow B, h : A \Rightarrow \mathbf{N} \Rightarrow B \Rightarrow B$, satisfying:

$$\begin{aligned} \mathcal{R}ahx0 &= ax \\ \mathcal{R}ahx(Sy) &= hxy(\mathcal{R}ahxy) \end{aligned}$$

Again we may ask: how do we know such primitive recursive definitions uniquely specify the intended function?

Such uniqueness questions make sense in many contexts: arithmetic theories [29], first-order term rewriting theories [9] , primitive recursive arithmetics [14], simply and higher-order typed lambda calculi and related functional languages[13, 20, 29], categorical programming languages [5, 15] and more generally wherever we define a procedure iteratively on an inductive data type[15, 16]. If mathematical induction is provided explicitly within the formal theory, the problem of uniqueness often becomes trivial (e.g.

[1]Research supported by Aids in Research of the Ministry of Education, Science and Culture of Japan, and the Oogata Kenkyu Josei grant of Keio University [2] Research supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada

Received by the editors 1999 March 8 and, in revised form, 1999 November 15.

Published on 1999 November 30.

Key words and phrases: typed lambda calculus, rewriting theory, strong normalization, Mal'cev operations.

[20], p.192.) But usual functional programming languages, term rewriting languages, or categorical languages do not explicitly have induction, hence the uniqueness issue is more problematic. We should mention that similar problems also appear in the theory of higher-order matching [10].

The question of (provable) uniqueness of terms also arises quite naturally from the viewpoint of category theory [20, 21]. For example, whenever we demand that arrows arise from an associated *universal mapping property*, then we are immediately led to the question of the (provable) uniqueness of terms in an appropriate language. Here we are concerned with the property of being a (strong) natural numbers object in a free cartesian closed category, equivalently the problem of uniqueness of iteration in Gödel's T [29, 13].

For many equational calculi, a simple-minded solution to the uniqueness problem involves adding a kind of extensionality or *uniqueness rule.* This rule says: "any term t satisfying the same defining equations as a given term, must be equal to that term." The uniqueness rule may be expressed in first order logic as a universally quantified conditional equation, possibly between terms of higher type [22] .

However, in many interesting cases (e.g. product and coproduct types) we can do much better: we can present uniqueness equationally and build a terminating (= strongly normalizing) higher-typed rewrite system for the theory [20]. [1] Here we use a technique of Lambek[18] involving Mal'cev operations to equationally define uniqueness of iterators (recursors) in Gödel's T. We then prove this lambda theory is undecidable; thus it is impossible to simultaneously have confluence (Church Rosser) and termination. In fact, we show that confluence fails, but give a proof of termination using a refined computability predicate argument (Section 4). This answers a question of J. Lambek. On the other hand, the theory is ground-CR (= Church-Rosser for closed terms of base type), thus it forms a natural, undecidable computation theory. This suggests the existence of natural, strongly normalizing functional languages which are Turing complete and could serve as the basis of interesting programming languages. Another consequence of this work is the undecidability of the coherence problem for cartesian closed categories with strong natural numbers objects. We end with a brief discussion of some extensions of these results, for example to the data type of Brouwer ordinals.

2. Typed Lambda Calculi

We use the usual explicitly typed lambda calculus (with recursors) and its notational conventions, cf. [13, 20, 29].

Types

Types are freely generated from a basic type **N** (for natural numbers) under the operation

[1]For typed lambda calculi with product types, uniqueness of pairing is given by the *surjective pairing* equation [20]. For coproduct types, the dual equation is obvious in a categorical language [20], Part I, Section 8. Curiously, the associated rewriting theory for typed lambda calculus with coproduct types is very difficult [11].

$\Rightarrow$.

Terms

For each type A, there is an infinite set of variables x_i^A of type A, as well as specified constants for primitive recursion, typed as follows:

$$0 : \mathbf{N} \qquad S : \mathbf{N} \Rightarrow \mathbf{N} \qquad \mathcal{R}_{A,B} : (A \Rightarrow B) \Rightarrow (A \Rightarrow \mathbf{N} \Rightarrow B \Rightarrow B) \Rightarrow A \Rightarrow \mathbf{N} \Rightarrow B$$

Terms are freely generated from the variables and constants by *application* and λ-*abstraction*, with the usual typing constraints. In shorthand, terms have the form

$$\mathit{variable} \mid \mathit{constant} \mid M`N \mid \lambda x : A.\varphi$$

where $M : A \Rightarrow B$, $N : A$. We usually write MN for $M`N$.

Equations

We identify terms up to change of bound variables (α-congruence). Equality is the smallest congruence relation closed under substitution, satisfying β, η and

$$\begin{aligned} \mathcal{R}_{A,B}ahx0 &= ax \\ \mathcal{R}_{A,B}ahx(Sy) &= hxy(\mathcal{R}_{A,B}ahxy) \end{aligned}$$

for all $a : A$, $h : A \Rightarrow \mathbf{N} \Rightarrow B \Rightarrow B, x : A, y : \mathbf{N}$.

Lambda Theories

Among standard extensions of lambda calculi to which our methods apply, we mention:

(a) Lambda calculi with product types and surjective pairing [20, 13]. Such systems are closely connected to *cartesian closed categories* (= ccc's). However for the purposes of constructing *free* ccc's, product types are unnecessary ([25]); cf Section 5 below. In the presence of product types with surjective pairing, the recursor $\mathcal{R}$ may be defined in terms of the simpler notion of iterator $\mathcal{I}_A : A \Rightarrow (A \Rightarrow A) \Rightarrow \mathbf{N} \Rightarrow A$ (cf [20], p. 259,[13], p. 90) The type $\mathbf{N}$ with iterators corresponds to the categorical notion of *weak natural numbers object* in ccc's [20].

    In this paper we shall use recursors $\mathcal{R}$ since (i) we do not necessarily assume product types, (ii) recursors are necessary for more general type theories (e.g. Girard's system $\mathcal{F}$) as well as for more general categorical frameworks (e.g. Lambek's multicategories, [19]).

(b) We may consider extensions of typed lambda calculus by adding *first-order logic* to the equational theory [23, 22] . This will be discussed in the next section. Finally, we may extend by additional data types. For example, in the last section we consider adding the data type of Brouwer ordinals, among others.

NOTATION: As usual, the type expression $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n$ abbreviates $A_1 \Rightarrow (A_2 \Rightarrow \cdots (A_{n-2} \Rightarrow (A_{n-1} \Rightarrow A_n)) \cdots)$, i.e. association to the right. Even if there are no product types, we sometimes abbreviate $A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n$ by the Curried expression $(A_1 \times \cdots \times A_{n-1}) \Rightarrow A_n$ but we still write terms in the usual way without pairing. The reader can easily add pairing to terms if there are genuine product types. The term expression $a_1 a_2 \cdots a_n$ abbreviates $(\cdots((a_1`a_2)`a_3) \cdots `a_{n-1})`a_n$, i.e. association to the left.

3. Uniqueness of Recursors

3.1. Natural Numbers Objects. In categorical logic [20], a *natural numbers object* (= NNO) in a cartesian closed category (= ccc) $\mathcal{C}$ is an object $\mathbf{N}$ together with arrows $\mathbf{1} \xrightarrow{0} \mathbf{N} \xrightarrow{S} \mathbf{N}$ which is initial among all diagrams of the shape $\mathbf{1} \xrightarrow{a} A \xrightarrow{g} A$. This means: for all arrows $a : \mathbf{1} \to A$ and $g : A \to A$ there is a *unique* iteration arrow $It_A(a, g) : \mathbf{N} \to A$ satisfying: (i) $It_A(a, g) \circ 0 = a$ and (ii) $It_A(a, g) \circ S = g \circ It_A(a, g)$. A *weak* natural numbers object $\mathbf{N}$ in a ccc $\mathcal{C}$ merely postulates the *existence*, but not uniqueness, of the iterator arrow above. In the typed lambda calculus associated to $\mathcal{C}$, a weak NNO is equivalent to having an iterator $\mathcal{I}_A : A \times (A \Rightarrow A) \times \mathbf{N} \Rightarrow A$ in the language (cf. [20], p. 70).

3.2. Lambek's Uniqueness Conditions. We now consider *uniqueness* of lambda terms defined by iterators or, more generally, recursors. Girard discusses the subtleties of such questions, and the defects of standard syntax, in [13], pp. 51, 91.

The use of first-order logic to strengthen lambda calculus is familiar, for example in discussing extensionality, lambda models, and well-pointedness of ccc's, cf. [22]. In a similar manner, uniqueness of It_A in ccc's can be guaranteed by the following conditional rule: given arrows $a : \mathbf{1} \to A$ and $g : A \to A$, if $f : \mathbf{N} \to A$ is *any* arrow satisfying $f \circ 0 = a$ and $f \circ S = g \circ f$ then $f = It_A(a, g)$. Let us translate this into lambda calculus.

More generally, we consider typed lambda calculi $\mathcal{L}$ presented with recursors $\mathcal{R}_{A,B}$. Given any terms a and h of appropriate type, uniqueness of $\mathcal{R}_{A,B}ah$ may be guaranteed by the following *uniqueness rule*: for any term $f : A \times \mathbf{N} \Rightarrow B$, and variables x, y of appropriate types:

$$(U_{f,h}) \qquad \frac{\forall x \forall y [fx(Sy) = hxy(fxy)]}{f = \mathcal{R}_{A,B}ah} \qquad \text{where } \ a = \lambda x : A.fx0$$

Here, for any x, the initial value of the recursion, $fx0$, is simply defined to be ax .

Lambek [18, 17] proved that it is possible to *equationally* represent the first-order rule $U_{f,h}$, provided we assume the existence of certain Mal'cev operators. Let us recall his proof.

Suppose all the types A in the lambda calculus $\mathcal{L}$ have a Mal'cev operation; i.e. a closed term $m_A : A^3 \Rightarrow A$ satisfying the following equations (in uncurried form):

$$m_A xxy = y \qquad m_A xyy = x$$

for all $x, y : A$. Fix a family $\{m_A \,|\, A \text{ a type}\}$ of such Mal'cev operations (we omit type subscripts). Consider the term H, where $Hmfh = \lambda xyz.m(hxyz)(hxy(fxy))(fx(Sy))$ for variables f, h (where f has the same type as $\mathcal{R}_{A,B}ah$ and $Hmfh$ has the same type as h), and the following equational rule:

$$(M_{f,h}) \qquad\qquad \mathcal{R}_{A,B}a(Hmfh) = f$$

NOTATION. From now on, when the context is clear, we shall omit type subscripts.

3.3. THEOREM. [Lambek([18])] *The following implications hold:*

(i) $M_{f,h}$ *implies* $U_{f,h}$

(ii) $U_{f,h'}$ *implies* $M_{f,h}$ *where* $h' = Hmfh$.

PROOF. In this proof, we freely use (η)-rule, i.e. extensionality. (i): Suppose $M_{f,h}$ and the hypotheses of $U_{f,h}$, i.e. suppose $fx(Sy) = hxy(fxy)$. Since m is a Mal'cev operator, $Hmfh = h$. Therefore by $M_{f,h}$, $fxy = \mathcal{R}a(Hmfh)xy = \mathcal{R}ahxy$, so $f = \mathcal{R}ah$, by (η), which is the conclusion of $U_{f,h}$.

(ii) Suppose $U_{f,h'}$. Note that the hypothesis of this rule is always true, since it says $fx(Sy) = h'xy(fxy) = m(hx(fxy))(hx(fxy))(fx(Sy)) = fx(Sy)$. Thus, the conclusion of the rule is true; so $f = \mathcal{R}ah'$, i.e. $M_{f,h}$, as required. ■

The theorem above expresses the sense in which we can replace the conditional uniqueness rules $U_{f,h}$ by the equations $M_{f,h}$. The problem of equationally defining the appropriate Mal'cev operators is addressed in section 3.2 below.

The rule $U_{f,h}$ above guarantees uniqueness of the recursor $\mathcal{R}_{A,B}ah$ for *given* a, h, which suffices for many purposes (e.g. in Lambek's work, it guarantees uniqueness of the iterator arrow in a strong natural number object). But it does *not* guarantee the uniqueness of the term $\mathcal{R}_{A,B}$ itself. The uniqueness of such $\mathcal{R}$ is a much stronger condition which we shall not discuss here.

One advantage of assuming the uniqueness rule $U_{f,h}$ is pointed out in [20], Proposition 2.9, p. 263: in simply typed lambda calculus with the uniqueness rule (equivalently, in the free ccc with NNO) *all primitive recursive functions and the Ackermann function may be defined by their usual free variable equations.* But in fact more is true: in the next proposition we show many familiar set-theoretic equations become provable for variables (not just for numerals). In what follows, $+, \cdot, \dot{-}$ are terms defined (using $\mathcal{R}$) by the usual primitive recursive equations on $\mathbf{N}$ (cf. [14]) .

3.4. PROPOSITION. [Goodstein[14], Lambek[18], Roman[27]] *In typed lambda calculus with the uniqueness rule* $U_{f,h}$*, the following identities hold for variables* $x, y : \mathbf{N}$

(i) $Sx \dot{-} Sy = x \dot{-} y$

(ii) $x \dot{-} x = 0$

(iii) $x \cdot (1 \dot{-} x) = 0$

(iv) $x \cdot (1 \dot{-} y) = x \dot{-} (x \cdot y)$

(v) $(x + y) \dot{-} y = x$

(vi) Associativity of $+$ *and* $\cdot$

(vii) $|x, y| = |y, x|$, *where* $|x, y| = (x \dot{-} y) + (y \dot{-} x)$

(viii) Commutativity of $+$ *and* $\cdot$

(ix) $x + (y \dot{-} x) = y + (x \dot{-} y)$

Proof. Detailed arguments are given in Goodstein [14] and/or Lambek[18] (cf. also Roman [27]). Re (ix), which is an important identity, either follow [14], 5.17 (p.108) or alternatively, note that both sides of (ix) (as functions of x, y) satisfy the following double recursion formula: $\varphi(0,0) = 0, \varphi(0, y+1) = y+1, \varphi(x+1, 0) = x+1, \varphi(x+1, y+1) = \varphi(x,y)+1$. Hence by uniqueness the equality follows, provided we can show φ is representable in typed lambda calculus (equivalently, in the free ccc with NNO). We omit the construction of φ except to remark it is similar to the proof that Ackermann's function is representable (cf. [1], p. 570,[20], p. 258). ■

Notation. We extend the arithmetical operations in the set $\mathcal{S} = \{0, +, \cdot, \dot{-}\}$ to all types by induction: (i) at type $\mathbf{N}$ the operations in $\mathcal{S}$ have their usual meanings. (ii) Suppose we know the operations in $\mathcal{S}$ at types A and B. Then we extend these operations to type $A \Rightarrow B$ by lambda abstraction, i.e. $0_{A \Rightarrow B} =_{def} \lambda x : A.0_B$, and $f \star_{A \Rightarrow B} g =_{def} \lambda x : A.fx \star_B gx$, for any operation $\star \in \{+, \cdot, \dot{-}\}$. Since all types have the form $A \equiv A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_{n-1} \Rightarrow \mathbf{N}$, an operation $a \star_A b$ is essentially $\lambda x_1 \cdots x_n . a x_1 \cdots x_n *_{\mathbf{N}} b x_1 \cdots x_n$, where $x_i : A_i$. Finally, we may now discuss the usual difference function in Goodstein at higher types: $|M, N| = (M \dot{-} N) + (N \dot{-} M)$

If we add explicit product types, we could also define the arithmetical operations $0_{A \times B}$ and $\star_{A \times B}$ "componentwise". We shall use the usual notation (without subscripts) for $+, \cdot, \dot{-}, 0$ at all types when the meaning is clear.

3.5. Proposition. *In typed lambda calculus with the uniqueness rule* $U_{f,h}$, *the following first-order schema holds for arbitrary terms* M, N *of the same type:* $\vdash \ |M, N| = 0 \leftrightarrow M = N$.

Proof. The $(\rightarrow)$ direction is the interesting one. Arguing informally, we prove the result for variables of type $\mathbf{N}$, the other cases following from appropriate lambda abstractions and substitutions. Suppose $|x, y| = 0$; then by Proposition 3.4(v, vii), $|x, y| \dot{-} (y \dot{-} x) = 0$, so $x \dot{-} y = 0$. Similarly, $y \dot{-} x = 0$. Hence, using equation Proposition 3.4(ix) above, we obtain: $x = x + (y \dot{-} x) = y + (x \dot{-} y) = y$. ■

3.6. Definable Mal'cev Operations. The rule $M_{f,h}$ is equational, provided the specification of the Mal'cev operations m_A are. In the pure typed lambda calculus, Lambek [19] showed that Mal'cev operations are actually definable from cut-off subtraction $\dot{-}$, provided we adjoin some simple equations (see below). Consider the inductively defined family of closed terms $m_A : A^3 \Rightarrow A$, uniquely determined (by functional completeness) by the following clauses in uncurried form:

1. $m_{\mathbf{N}}xyz = (x+z)\dot{-}y$, for variables $x, y, z : \mathbf{N}$.
2. $m_{A\Rightarrow B}uvw = \lambda x : A.\, m_B(ux)(vx)(wx)$, for variables $u, v, w : A \Rightarrow B$.

Remark. If typed lambda calculus is extended with products and terminal type, we may extend m_A (using explicit tupling) as follows: $m_{\mathbf{1}}\langle x, y, z\rangle = *$, for variables $x, y, z : \mathbf{1}$ and $m_{A\times B} = \langle m_A \circ \langle p_{A1}, p_{A2}, p_{A3}\rangle, m_B \circ \langle p_{B1}, p_{B2}, p_{B3}\rangle\rangle$, where p_{Ai} denotes the projection onto the ith-factor of the A component, etc.

To say $m_{\mathbf{N}}xyz = (x+z)\dot{-}y$ gives a Mal'cev operator at type $\mathbf{N}$ means we must be able to prove $(x+z)\dot{-}x = z$ and $(x+y)\dot{-}y = x$ for *variables* x, y, z. Alas, in the pure theory these equations are only provable for closed *numerals* $0, S0, S^2 0, \cdots$: to guarantee the result for variables we must postulate it, as follows:

3.7. Theorem. *Let $\mathcal{L}$ be pure typed lambda calculus (all types freely generated from $\mathbf{N}$). The inductively-defined family of terms $\{m_A \mid A \text{ a type}\}$ given above defines Mal'cev operations at each type, provided that at type $\mathbf{N}$ we add the axioms: $(x+z)\dot{-}x = z$ and $(x+y)\dot{-}y = x$, for variables x, y, z.*

Proof. By direct calculation. ∎

3.8. Uniqueness, Induction, and Undecidability. We now show that the uniqueness rule is in fact equivalent to a version of quantifier-free induction. We shall then give proofs of the undecidability of $\mathcal{L}$ with uniqueness.

3.9. Lemma. *The following are equivalent for any type B:*

1. *Uniqueness rule: for any closed term $f : A \Rightarrow (\mathbf{N} \Rightarrow B)$*

$$(U_{f,h}) \qquad \frac{\forall x \forall y [fx(Sy) = hxy(fxy)]}{f = \mathcal{R}_{A,B}ah} \qquad \textit{where} \quad a = \lambda x : A.fx0$$

2. *Goodstein Induction at type B: for any closed term $f : A \Rightarrow \mathbf{N} \Rightarrow B$*

$$(\textit{Goodstein Ind.}) \qquad \frac{\forall x \forall y [fx0 = 0 \quad (1\dot{-}fxy)\cdot fx(Sy) = 0]}{f = \lambda xy.0}$$

where $x : A$ and $y : \mathbf{N}$.

3. *The induction rule : for any closed terms $F, G : A \Rightarrow \mathbf{N} \Rightarrow B$*

$$(\textit{Induction}) \qquad \frac{\forall x, y.\;\; Fx0 = Gx0 \quad \begin{array}{c}[Fxy = Gxy] \\ \vdots \\ Fx(Sy) = Gx(Sy)\end{array}}{F = G}$$

where $x : A$ and $y : \mathbf{N}$ in the antecedent are universally quantified.

Proof. We show (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1), in each case by induction on the type B.

(a). $(U_{f,h}) \Rightarrow$ (Goodstein induction) : this is essentially Goodstein's argument, [14], p.35, using the identities of Proposition 3.4 (this proposition assumes $U_{f,h}$.) For completeness, we sketch the proof.

Case 1. $B = \mathbf{N}$. Let $f : A \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}$ be given satisfying the hypotheses of Goodstein Induction. Then

$$(1\dot{-}fxy)\cdot(1\dot{-}fx(Sy)) \;=\; (1\dot{-}fxy)\dot{-}((1\dot{-}fxy)\cdot fx(Sy)) \;=\; 1\dot{-}fxy \tag{1}$$

Define $g : A \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}$ by primitive recursion as follows (where $x : A$ and $y : \mathbf{N}$ are variables): $gx0 = 1$, $gx(Sy) = gxy\cdot(1\dot{-}fxy)$. Let $hxy = gx(Sy)$. Then using equation (1)

$$hx(Sy) \;=\; gx(Sy)\cdot(1\dot{-}fx(Sy)) \;=\; gxy\cdot(1\dot{-}fxy) = gx(Sy) \tag{2}$$

Also $hx0 = 1 = gx0$. An easy argument ([14], p.34) then shows $hxy = gxy$. Hence, $gx0 = 1$ and $gx(Sy) = gxy$. By [14], 2.7304 (p. 55), $g = \lambda xy.1$. In particular, $1\dot{-}fxy = 1$. Hence $fxy = fxy\cdot(1\dot{-}fxy) = 0$, by Proposition 3.4, (iii) .

Case 2: In general, any type has the form $B = A_1 \Rightarrow A_2 \cdots \Rightarrow A_{n-1} \Rightarrow \mathbf{N}$. Thus a closed term $f : A \Rightarrow \mathbf{N} \Rightarrow B$ means $f : A \Rightarrow \mathbf{N} \Rightarrow A_1 \Rightarrow A_2 \cdots \Rightarrow A_{n-1} \Rightarrow \mathbf{N}$. Thus $f = \lambda x^A y^{\mathbf{N}} z_1^{A_1} \cdots z_{n-1}^{A_{n-1}}.fxy\vec{z_i}$, where $fxy\vec{z_i} : \mathbf{N}$. Now apply the same argument as in Case 1 (with extra parameters $\vec{z_i}$).

(b). (Goodstein Induction) $\Rightarrow$ (Induction):

Case 1: $B = \mathbf{N}$. Let Φ be the closed term $\lambda xy.|Fxy, Gxy|$. By Proposition 3.5, Induction is equivalent to saying:

$$\cfrac{\forall x, y.\ \ \Phi x0 = 0 \quad \begin{matrix}[\Phi xy = 0]\\ \vdots \\ \Phi x(Sy) = 0\end{matrix}}{\Phi = 0}$$

The proof that Goodstein Induction implies the above rule is (slightly) modified from Goodstein [14], p.109. Note that Goodstein's proof assumes terms $F, G : \mathbf{N} \Rightarrow \mathbf{N}$; the same proof goes through verbatim if we assume there is an extra parameter y of type A, i.e. that F, G are actually of type $A \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}$.

Case 2: $B = A_1 \Rightarrow A_2 \cdots \Rightarrow A_{n-1} \Rightarrow \mathbf{N}$. Hence, as above, the problem reduces to the case $B = \mathbf{N}$, with extra parameters $z_i : A_i$.

(c). (Ind.) $\Rightarrow$ $(U_{f,h})$: The following proof is valid for all types B. Suppose the hypotheses of U_{fh}. Let $F = \lambda xy.fxy$ and $G = \lambda xy.\mathcal{R}ahxy$. We wish to prove $F = G$, i.e. $Fxy = Gxy$, for all x, y. Clearly $Fx0 = fx0 = ax = \mathcal{R}ahx0 = Gx0$. Now suppose $Fxy = Gxy$, i.e. $fxy = \mathcal{R}ahxy$. Then, using the hypothesis of U_{fh} ,

$$Fx(Sy) = fx(Sy) = hxy(fxy) = hxy(\mathcal{R}ahxy) = \mathcal{R}ahx(Sy) = Gx(Sy) \;\; .$$

So by Induction, $F = G$. ■

From the availability of induction in the simply typed lambda calculus with uniqueness, it follows that the first-order presentation is undecidable. But there are direct proofs for the equational presentation as well:

3.10. Theorem. *The typed lambda calculus $\mathcal{L}$ with uniqueness $U_{f,h}$ is undecidable. Hence we cannot simultaneously have SN and CR for any associated rewriting system.*

Proof. Observe that with uniqueness we can represent polynomials with positive integer coefficients (cf the remarks before Proposition 3.4). Thus, from the undecidability of Hilbert's 10th problem, the general question of deciding if two such lambda terms are equal or not is undecidable. The conclusion about rewriting systems is familiar. SN and CR imply the word problem is decidable: to decide if two terms are equal or not, reduce them using SN to their unique normal forms, then check if the normal forms are identical (up to change of bound variables). ∎

Let us end this section with a few remarks.

(i) A related result by G. Dowek [10] yields the undecidability of pattern matching in lambda calculi supporting inductive types (as well as other theories).

(ii) Simply typed lambda calculus with product types and uniqueness $\mathcal{L} + U_{f,h}$ is very close to Troelstra's theory qf-λ-E-HA$^{\omega}_{p}$ of *the quantifier-free part of extensional arithmetic in all higher types, lambda operator and product types with surjective pairing* [29], p. 46, 62 (cf. the form of induction in the Lemma above.)

We can also give an indirect proof of undecidability based on Troelstra's theory; note, in particular, our calculus satisfies the ξ-rule: $s = t \Rightarrow \lambda x.s = \lambda x.t$. We follow Troelstra's proof [29], p. 62 that $\mathcal{S} = \{\langle\ulcorner m\urcorner, \ulcorner n\urcorner\rangle \mid\ \vdash m = n\}$ and $\mathcal{S}' = \{\langle\ulcorner m\urcorner, \ulcorner n\urcorner\rangle \mid\ \vdash m \neq n\}$ are recursively inseparable. Note, however, that Troelstra's proof also requires ([29], pp. 51-59) coding various equations of primitive recursive arithmetic, as well as simultaneous recursion, all proved using induction. Our proofs of similar equations in Proposition 3.4 used the uniqueness axiom $U_{f,h}$ instead.

4. Rewriting theory for Uniqueness: SN and CR

Let $\mathcal{L}$ be the typed lambda calculus with primitive recursors and Mal'cev operators, considered as a rewrite system[16, 20, 13]. To this end we orient all basic equations as usual, i.e. from left-hand side to right-hand side, abbreviated LHS $\gg$ RHS . In particular, we orient in this manner the basic Mal'cev equations at type $\mathbf{N}$: $mxxy \gg x$, $mxyy \gg y$, the equations required for defining Mal'cev operators (see Proposition 3.7), as well as β and η.

4.1. Proposition. *The system $\mathcal{L}$ of typed lambda calculus, recursors (without uniqueness) and Mal'cev equations considered as a rewriting theory oriented as above satisfies SN and CR.*

The proof is a corollary of the work of Jouannaud-Okada[16].

$$\begin{tikzcd}
 & Ra(Hmfh) \arrow[dl, "\gg"'] \arrow[dr, "\gg"] & & & Ra(Hmfh)xy \arrow[dl, "\gg"'] \arrow[dr, "\gg"] & \\
f & & Rah & fxy \arrow[d] & & Rahxy \\
 & & & 0 & &
\end{tikzcd}$$

Figure 1: Failure of CR

We now turn to the Mal'cev uniqueness rule. Consider the following two basic orientations:

1. *Reduction* $Ra(Hmfh) \gg f$

2. *Expansion* $f \gg Ra(Hmfh)$

Let $\mathcal{L}_{red}$ = the rewriting theory obtained from $\mathcal{L}$ with the above reduction rule and $\mathcal{L}_{exp}$ = the rewriting theory obtained from $\mathcal{L}$ with the above expansion rule.

4.2. Proposition. *The system $\mathcal{L}_{exp}$ is CR but not SN.*

Proof. Since the LHS of the expansion rule is a (higher-type) variable and the original system $\mathcal{L}$ is CR, any critical pair is joinable by 1-step parallel reduction, [2] Hence, the system is CR. Obviously, the system $\mathcal{L}_{exp}$ with the expansion rule above is *not* SN, since the RHS expands with each reduction. ∎

The system $\mathcal{L}_{red}$ is considerably more interesting. In what follows we show that $\mathcal{L}_{red}$ is SN, not CR, but is ground CR (i.e. CR for *closed* terms of ground type).

4.3. Proposition. *The system $\mathcal{L}_{red}$ is not CR. Indeed, it is not even CR for open terms of ground type.*

Proof. Consider $f = \lambda xy.0$ and $h = \lambda xyz.0$, where f is of the type of the recursor $\mathcal{R}ah$. Consider the following critical pair between the reduction rule and the Mal'cev rule:

(i) $Ra(Hmfh) \gg f$, (ii) $Ra(Hmfh) \gg Rah$

Reduction (i) comes from our orientation of the Mal'cev equation $M_{f,h}$. Reduction (ii) arises as follows: since $hxy(fxy) = 0 = fx(Sy)$, then

$$Hmfh = \lambda xyz.m(hxyz)(hxy(fxy))(fx(Sy)) \gg \lambda xyz.0 \equiv h$$

where f and Rah are irreducible. Note that all other critical pairs are actually joinable. See Figure 1(i). In Figure 1(ii), we see a modification of (i) where $Rahxy : \mathbf{N}$ is an open term with variables $x : A, y : \mathbf{N}$. □ ∎

4.4. Proposition. *The system $\mathcal{L}_{red}$ is ground CR, i.e. CR for closed terms of base type.*

Proof. The only important case is shown by diagram (ii) of Figure 1, where we substitute *closed* terms $\hat{x} : A$ and $\hat{y} : \mathbf{N}$ for the variables x, y, respectively. Assuming $\mathcal{L}_{red}$ is SN (see below), then by an inductive argument ([29], p. 104) one shows $\vdash \hat{y} = S^n 0$ (for some $n \in \mathbf{Z}^+$) or $\vdash \hat{y} = 0$. In either case, we note that (ii) is joinable in either one or two steps, with $Rah\hat{x}\hat{y} \gg^* 0$. ∎

Strong Normalization of $\mathcal{L}_{red}$

4.5. Theorem. *The system $\mathcal{L}_{red}$ is SN, i.e. if t is an arbitrary term, then every reduction path starting from t halts (at an irreducible).*

The proof uses the Tait computability method ([29]). For technical reasons it does not suffice to directly define computability for $\mathcal{L}_{red}$, but rather we must introduce an auxiliary language $\mathcal{L}'_{red}$ (see Remark 4.8 below) [2].

The language $\mathcal{L}'$ is $\mathcal{L}$ but with a kind of "parametrized" Mal'cev operator $m'_A : A^3 \times \mathbf{N} \Rightarrow A$ defined as follows:

1. $m'_{\mathbf{N}}xyzw = (x + z) \dot{-} y$ for variables $x, y, z : \mathbf{N}$
2. $m'_{A \Rightarrow B}fghw = \lambda x : A\lambda y : \mathbf{N}.m'_B(fx)(gx)(hx)y$ for variables $f, g, h : A \Rightarrow B$, $w : \mathbf{N}$.

Note that for every type A, variables $x, y, z : A$, $w : \mathbf{N}$, the two defined Mal'cev operators are indistinguishable, in the sense that: $\vdash m'_A xyzw = m_A xyz$, i.e. m' is essentially m with an extra dummy variable w.

In $\mathcal{L}'$, analogously to $\mathcal{L}$, we introduce the term

$$H'm'fh = \lambda xyz\vec{p}.m'(hxyz\vec{p})(hxy(fxy)\vec{p})(fx(Sy)\vec{p})(fxy\vec{p})$$

where $\vec{p}$ is a sequence of variables such that $fxy\vec{p}$ is of type $\mathbf{N}$.

We now form the lambda calculus $\mathcal{L}'_{red}$ completely analogously to $\mathcal{L}_{red}$, but with $M_{f,h}$ replaced by the reduction

$$(M'_{fh}) \qquad\qquad R_{A,B}a(H'm'fh) \gg f$$

4.6. Theorem. *The system $\mathcal{L}'_{red}$ is SN, i.e. if t is an arbitrary term, then every reduction path starting from t halts (at an irreducible).*

The proof is a modification of the well-known Tait computability method [29]. One first defines the *computable terms of type A*, $Comp_A$, by induction on types as in [29]:

$$t \in Comp_A \quad \Leftrightarrow \quad t \in \mathbf{SN} \;\text{ for atomic types } A. \tag{3}$$

$$t \in Comp_{A \Rightarrow B} \quad \Leftrightarrow \quad \forall a(a \in Comp_A \Rightarrow t`a \in Comp_B). \tag{4}$$

[2]Known SN techniques do not suffice. For example, in order to adapt Blanqui, Jouannaud, Okada[3] f in the Mal'cev uniqueness rule must be an *accessible subterm*, which is not generally true if f is of higher type. In the earlier paper of Jouannaud and Okada [16] the SN techniques do not apply since the higher type variable h on the LHS disappears on the RHS.

One now proves the following by induction on types:

CR1. If t is computable, then t is SN.
CR2. If t is computable and $t \gg^* t'$ then t' is computable.

We now need to prove a lemma (by induction on terms): *All terms are computable* , from which the main theorem follows by CR1. In the proof of the lemma, one needs a stronger inductive hypothesis to handle the case of lambda abstraction. This is summarized by the following (cf. [13], p. 46) :

4.7. Lemma. *Let $\mathcal{L}'_{red}$ be simply typed lambda calculus with reduction on the new Mal'cev operator, as above. If $t(x_1, ..., x_n) : B$ is any term with free variables $x_i : A_i$, and if the terms $a_i : A_i$ are all computable, then $t(a_1, ..., a_n)$ is itself computable. In particular, since variables are computable, any term $t(x_1, ..., x_n) : B$ is computable.*

Proof. The proof is on the complexity of t. We consider the substitution instance $t[\vec{d}/\vec{x}]\vec{b}$, where the d_i are computable, and the b_j are also computable so that $t[\vec{d}/\vec{x}]\vec{b}$ is of type $\mathbf{N}$. It suffices to verify that this term is SN. All cases proceed as usual, except the critical case when $t[\vec{d}/\vec{x}]\vec{b}$ is $Ra(H'm'fh)\vec{b'}$, the redex of the Mal'cev reduction (M'_{fh}). We show that after (M'_{fh}), $f\vec{b'}$: $\mathbf{N}$ is SN. There are essentially two key forms of such t : (i) either $t \equiv R\hat{a}(H'm'\hat{f}\hat{h})$ (so $\hat{f}[\vec{d}/\vec{x}] = f$ (for the f in $Ra(H'm'fh)$)) or (ii) t is not of the form (i); then f must be a proper subterm of some d_i or b_j. Case (i) is very easy to prove: by induction hypotheses, since $\hat{f}$ is a subterm of t, $\hat{f}[\vec{d}/\vec{x}]$ is computable, so f is computable, so $f\vec{b}$ is SN. For case (ii) $H'm'fh$ is contained in either some d_i or some b_j. Hence, since these are SN, $H'm'fh$ is SN. In particular *using the chosen formula for H' in terms of m'* the subterm $fxy\vec{p}$ is strongly normalizable for any computable terms substituted for $x, y, \vec{p}$; in particular $f\vec{b'}$ is SN. ■

4.8. Remark. The reader should note that in the proof above, the key fact that H' contains the subterms $fxy\vec{p}$ is critical–the original Mal'cev rule, using $Hmfh$, will not directly work.

Since the tree of reductions of m is a subtree of that of m', we obtain:
Corollary. (Theorem 4.5) *$\mathcal{L}_{red}$ is SN.*

4.9. Remark. Although we do not assume genuine product types here, they may be accounted for, just as in the setting of Jouannaud-Okada [16]. Products use the Tait-Girard computability method [13], Chap. 6 (cf. also [20], pp. 88-92)), along with an auxiliary notion of "neutral term" and an extended computability predicate satisfying CR3 (cf. [13]). All this extends to the setting here.

5. Applications and Extensions

5.1. Coherence for CCC's. As an immediate corollary of the above results, we obtain:

5.2. Example. The undecidability of the word problem (Coherence Problem) for equality of arrows in the free cartesian closed category with (strong) NNO generated by the empty graph.

Here, the free ccc with strong NNO $\mathcal{F}$ can be presented in several ways from term models:

(i) Following Pitts [25, 8] (4.2.1) we may construct $\mathcal{F}$ from simply typed lambda calculus without product types (e.g. our language $\mathcal{L}$ above) as follows: the objects of $\mathcal{F}$ will be finite lists of types; arrows will be equivalence classes of finite lists of terms in context. The Mal'cev equations guarantee that the NNO is strong.

(ii) Following Lambek and Scott [20], we may construct $\mathcal{F}$ using our language $\mathcal{L}$ with product and terminal types; here objects are types, and arrows are equivalence classes of terms with at most one free variable. Again, the Mal'cev equations guarantee that the NNO is strong.

In either case above we know the equational theory so generated is undecidable, so equality of arrows is too (for the Pitts' construction use Theorem 3.10; for the Lambek-Scott, we use the extension of Theorem 3.10 to product types (cf. Remark 4.9).

Remarks:

1. Gödel's $\mathcal{T}$ (i.e. simply typed lambda calculus with recursors) is known to be decidable, since it is CR and SN. We wish to emphasize that the extended theory with Uniqueness $\mathcal{L}_{red}$ is undecidable because of failure of Church-Rosser, *not* because of failure of SN.

2. We should like to emphasize that simply typed lambda calculus with uniqueness of iterators (as an equational theory) is a mathematically natural extension of $\mathcal{T}$, containing a purely equational equivalent to induction, and closely related to familiar theories of arithmetic of finite types .

5.3. Example. The equational theory $\mathcal{L}_{red}$ of typed lambda calculus with Mal'cev's equations is a natural example of an SN, non-CR, ground-CR, undecidable rewriting system.

Natural examples of such theories are not common. Such a theory can provide a computational model for functional languages, since computation of terms of ground type (by normalization) gives unique values. Moreover, $\mathcal{L}_{red}$ is undecidable, unlike most SN, CR typed lambda calculi which only represent a proper subclass of the total recursive functions.

$\mathcal{L}_{red}$ can also be used for verification of inductive properties: the uniqueness rule for terms defined by recursion is equivalent to an induction rule, and hence could be used as an alternative proof technique to traditional inductive arguments. We would like to see how these techniques could apply to inductive theorem proving and related decision problems in rewriting theory (cf. [6]).

5.4. Extension to Brouwer Ordinals. We shall now sketch how to extend the treatment to a larger class of data types. Such data types satisfy a domain equation $\alpha \cong A \Rightarrow \alpha$, where the type variable α appears covariantly; in particular, we consider $\alpha \notin A$ (cf. [20, 15]). To illustrate, we consider the data type of Brouwer Ordinals added to the simply type lambda calculi $\mathcal{L}$ above.

5.5. Definition. The type of *Brouwer ordinals Ord* is a type with distinguished constants: $0 : Ord, S : Ord \Rightarrow Ord,\ lim : (\mathbf{N} \Rightarrow Ord) \Rightarrow Ord.$ The recursor $\mathcal{R}$ is defined on Ord by postulating the following rules:

$$\begin{aligned} \mathcal{R}ahgx0 &= ax && (5)\\ \mathcal{R}ahgx(Sy) &= hxy(\mathcal{R}ahgxy) && (6)\\ \mathcal{R}ahgx(lim\ k) &= gxk(\lambda z(\mathcal{R}ahgx(kz))) && (7) \end{aligned}$$

Here: $a : A \Rightarrow Ord,\ h : A \Rightarrow \mathbf{N} \Rightarrow Ord,\ g : A \Rightarrow ((\mathbf{N} \Rightarrow Ord) \Rightarrow (\mathbf{N} \Rightarrow Ord))$, and $x : A$. For simplicity we are assuming $\mathcal{R}ahgx0$ is of type Ord, but it could be of any higher type.

We wish to remark that this definition is merely the *data type* of Brouwer ordinals, and has nothing to do with set-theoretic transfinite recursion.

We extend our analysis of Lambek's uniqueness conditions in Section 3 as follows. The rule $U_{f,h,g}$ now has two premisses:

$$(U_{f,h,g}) \qquad \frac{\forall x\forall y[\ fx(Sy) = hxy(fxy)\ ] \quad \forall xk[\ fx(lim\ k) = gxk(\lambda z(fx(kz)))\ ]}{f = \mathcal{R}ah}$$

where $a = \lambda x : A.fx0$. Similarly, we have

$$(M_{f,h,g}) \qquad \mathcal{R}a(Hmfh)(H'mfg) = f$$

Here, $Hmfh$ is the same as before, while

$$H'mfg =_{def} \lambda xku[m(gxku)(gxk(\lambda z(fx(kz))))(fx(lim\ k))].$$

The following is analagous to Lambek's Theorem 3.3, in Section 3.

5.6. Theorem. *The following implications hold:*

(i) $M_{f,h,g}$ *implies* $U_{f,h,g}$

(ii) $U_{f,h',g'}$ *implies* $M_{f,h,g}$ *where* $h' = Hmfh$ *and* $g' = H'mfg$ *.*

Proof. Re (i), assume the two premises of $U_{f,h,g}$. In particular, from the second premise, $fx(limk) = gxk(\lambda z(fx(kz)))$. Therefore, by the property of the Mal'cev operator m and by the definition of $H'mfg$, we have $H'mfg = \lambda xku(gxku) = g$. In the same way, from the first premise, we get $Hmfh = h$. Therefore, from $M_{f,h,g}$, $\mathcal{R}ahg = f$. This proves (i).

Re (ii), below we prove the following two equations:

$$\begin{aligned} fx(Sy) &= h'xy(fxy) && (8)\\ fx(limk) &= g'xk(\lambda z(fx(kz))). && (9)\end{aligned}$$

By using equations (8) and (9) as the two premisses of $U_{f,h',g'}$, then by $U_{f,h',g'}$ $f = Rah'g'$, namely, $f = Ra(Hmfh)(H'mfg)$, i.e., $M_{f,h,g}$ holds.

Since the proof of (8) is the same as the type $\mathbf{N}$ case in the original Lambek theorem, we prove (9) only: $g'xk(\lambda z(fx(kz))) =_{def} (H'mfg)xk(\lambda z(fx(kz))) = m(gxk(\lambda z(fx(kz))))(gxk(\lambda z(fx(kz))))(fx(lim\ k)) = fx(lim\ k)$, as required. ■

As for the rewriting theory of this *Ord* system, the analog of Proposition 4.1, that the combined system of first-order Mal'cev rules and the *Ord*-recursor rules is SN follows from a recent theorem of Blanqui, Jouannaud, and Okada [3]. Letting again $\mathcal{L}_{red}$ denote this new language with the reduction ordering, $\mathcal{L}_{red}$ is not CR but is ground CR by the same proof as Proposition 4.3. The analog of Theorem 4.5 establishing SN similarly applies to this extended language.

Finally, concerning the discussion in Lemma 3.9 on the equivalence of uniqueness and various forms of induction, the situation is a little more complex here. First we observe that the *Ord*-induction rule is:

$$(Ord - Ind)\qquad \frac{Fx0 = Gx0 \qquad \begin{array}{c}[Fxy = Gxy]\\ \vdots\\ Fx(Sy) = Gx(Sy)\end{array} \qquad \begin{array}{c}[\ \forall z(Fx(fz) = Gx(fz)\ ]\\ \vdots\\ Fx(lim\ f) = Gx(lim\ f)\end{array}}{F = G}$$

where y and f are eigen variables.

In order to naturally extend the equivalence in Lemma 3.9, it is necessary to discuss the computations used in Goodstein Induction in this more general framework. For this, it appears necessary to set up a primitive recursive ordinal notation system to represent $+, \dot{-}, m$, etc. in order to simulate (in our now more general system) the basic properties used in the standard $\mathbf{N}$-case (cf. also the discussion after Theorem 3.6). This appears routine, but we omit the detailed verification here.

Finally, we remark that from a categorical viewpoint, the datatype *Ord* makes sense in any ccc with natural numbers object N: it is an object *Ord* with arrows $\mathbf{1} \to Ord, S : Ord \to Ord, lim : Ord^N \to Ord$ satisfying the appropriate equations (analogous to a parametrized natural numbers object [20, 27])

5.7. Remark. Concerning more general inductive data types (cf. [7, 3]) the above results can be extended under the condition that we can define Mal'cev operations on the data type. In the examples we know, this involves defining some analog of $\dot{-}$ (or, for multiplicative operations, $(-)^{-1}$). This is not always easy to do. For example, for the data type of lists of integers, $\ell ist(\mathbf{N})$, there is no obvious choice for $\dot{-}$. On the other hand, the data type of lists of length k, $\ell ist_k(\mathbf{N})$, does have a Mal'cev operator using pointwise subtraction: $(a_1, \cdots, a_k)\dot{-}(b_1, \cdots, b_k) = (a_1\dot{-}b_1, \cdots, a_k\dot{-}b_k)$.

Another way to extend these results is simply to postulate a Mal'cev operator, and the associated rewriting rule, at each type.

References

[1] H. P. Barendregt. *The Lambda Calculus*, North-Holland, 1984.

[2] J. Bergstra and J. W. Klop, Conditional rewrite rules; confluence and termination. *J. Comput. Systems Sci.* 32 (1986) pp. 323-362.

[3] Blanqui, J-P. Jouannaud, and M. Okada. Calculus of Inductive Constructions, in: RTA99 (Rewriting Theory and Applications), *Springer LNCS* 1631, pp. 301-316.

[4] V. Breazu-Tannen and J. Gallier, Polymorphic rewriting conserves algebraic normalization, *Theoretical Computer Science*, **83** (1), 1991, pp. 2-28.

[5] J.R.B. Cockett and D. Spencer. Strong Categorical Datatypes I, Category Theory 1991, Proc. Summer Category Theory Meeting, Montreal, CMS Conference Proceedings v.13, Can. Math. Soc. (ed. by R.A.G. Seely), 1992.

[6] H. Comon, P. Narendran, R. Nieuwenhuis, M. Rusinowitch. Decision Problems in Ordered Rewriting, *13th Annual IEEE Symposium on Logic in Computer Science (LICS)*, 1998, pp. 276-286.

[7] T. Coquand and C. Paulin-Mohering, Inductively defined types. In P. Martin-Löf and G. Mints, eds. *Proc. Colog '88*, LNCS 417, Springer 1990, pp. 50-66

[8] D. Čubrić, P. Dybjer and P.J.Scott, Normalization and the Yoneda Embedding, *Math. Structures in Computer Science*, Camb. U. Press, **8**, No.2, 1998, pp. 153-192.

[9] N. Dershowitz and J.-P. Jouannaud, Rewrite Systems, *Handbook of Theoretical Computer Science*, Chapter 15, North-Holland, 1990.

[10] G. Dowek, The undecidability of pattern matching in calculi where primitive recursive functions are representable, *Theoretical Computer Science* 107 (1993) pp. 349-356.

[11] N. Ghani, $\beta\eta$-equality for coproducts. *Typed Lambda Calculi and Applications*, SLNCS 902, 1995, pp. 171-185.

[12] J-Y. Girard, Linear Logic, *Theoretical Computer Science*, **50**, 1987.

[13] J-Y. Girard, Y.Lafont, and P. Taylor. *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science, v. **7**, Cambridge University Press, 1989.

[14] R. L. Goodstein, *Recursive Number Theory*, North-Holland Publishing Co., 1957.

[15] T. Hagino. *A categorical programming language*, Phd Thesis, Univ. of Edinburgh (1987).

[16] J-P. Jouannaud and M. Okada, Abstract Data Type Systems, *Theoretical Computer Science* **173** (1997), pp. 349-391. This is a revised version of an extended abstract by the same authors: A computation Model for Executable Higher-Order Algebraic Specification Languages, in *Logic in Computer Science* (LICS), 1991, Amsterdam, IEEE Computer Soc. Press, pp. 350-361. .

[17] J. Lambek, Cartesian closed categories and typed λ-calculi, in: Cousineau, Curien, and Robinet (eds.), Combinators and functional programming languages, SLNCS **242** (1986), pp. 136-175.

[18] J. Lambek, On the Unity of algebra and Logic, SLNM 1348, *Categorical Algebra and its applications,* F. Borceux, ed. Springer-Verlag, 1988.

[19] J. Lambek, Multicategories Revisited. *Contemp. Math.***92**, 1989, pp. 217-239.

[20] J.Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, Cambridge Studies in Advanced Mathematics **7**, Cambridge University Press, 1986.

[21] S. Mac Lane. *Categories for the Working Mathematician*, Graduate Texts in Mathematics 5, Springer-Verlag, 1971.

[22] J. C. Mitchell and P. J. Scott,Typed Lambda Models and Cartesian Closed Categories, *Contemp. Math.* **92**, 1989, pp. 301-316.

[23] J. C. Mitchell, *Foundations for Programming Languages*, MIT Press, 1996.

[24] M. Okada, Strong normalizability of the combined system of simply typed lambda calculus and an arbitrary convergent term rewriting system, *Int. Symp. on Symbolic and Algebraic Computation 89*, ACM, 1989.

[25] A. M. Pitts , Categorical Logic, *Handbook of Logic in Computer Science* S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, eds. Oxford University Press, 1996, Vol. 6 (to appear)

[26] D. Prawitz. *Natural Deduction* , Almquist & Wilksell, Stockhom, 1965.

[27] L. Roman, Cartesian Categories with Natural Numbers Object, *J. Pure and Applied Algebra* **58**(1989), pp. 267-278.

[28] S. Stenlund, *Combinators, lambda terms, and proof theory*, D. Reidel, 1972.

[29] A. S. Troelstra. *Metamathematical investigations of intuitionistic arithmetic and analysis*, Springer LNM 344, 1973.

[30] B. Werner, Une théorie des constructions inductives, Thèse de doctorat, U. Paris 7, 1994.

Department of Philosophy, Keio University,
2-15-45 Mita, Minatoku, Tokyo,
Japan, 108
and
Department of Mathematics, University of Ottawa,
585 King Edward, Ottawa, Ont.,
Canada K1N6N5

Email: `mitsu@abelard.flet.mita.keio.ac.jp` and `phil@mathstat.uottawa.ca`

This article may be accessed via WWW at `http://www.tac.mta.ca/tac/` or by anonymous ftp at `ftp://ftp.tac.mta.ca/pub/tac/html/volumes/6/n4/n4.{dvi,ps}`