On seminormal expressions in lambda typed lambda calculus

M 19

On seminormal expressions in λ-typed λ-calculus

by

N. G. de Bruijn

April 1984

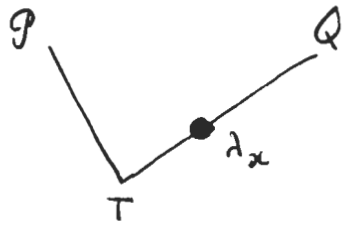
Department of Mathematics and Computer Science
Eindhoven University of Technology
PO Box 513, 5600 - MB EINDHOVEN
The Netherlands.

1. Introduction

This note might be considered as a continuation of an earlier note [1], where the expressions of λ-typed λ-calculus are presented in tree form (but will actually be independent of [1].) (AT $\lambda\tau$ - trees), In the present note we shall mainly use the linearized form, which is obtained from the tree-shaped form by representing

as $\langle \mathcal{P} \rangle\ Q$

and (using name-carrying lambda notation)



as $[x : \mathcal{P}]\ Q$

like in standard AUTOMATH notation (cf. [2]).

In such a formula we refer to a fragment of the form $\langle \mathcal{P} \rangle\ [x : \mathcal{R}]$ as an <u>AT-pair</u>.

We prefer to have formulas where there are never two applicators in front of an abstractor, like in

$$\langle \mathcal{P} \rangle\ \langle Q \rangle\ [x : \mathcal{R}]$$

If this does not happen, the expression has the form of a sequence of items which are either abstractors or AT-pairs, followed by a sequence of applicators, followed by a variable or by τ, like in

$$[t:T]\,\langle P_1\rangle[x:Q_1]\,\langle P_2\rangle[y:Q_2]\,\langle R_1\rangle\langle R_2\rangle\, y \qquad (1)$$

An expression will be called <u>seminormal</u> if this situation holds for all sub-expressions, sub-sub-expressions etc. as well. Or, slightly more precisely, we require that the expression is like (1), where T, P_1, Q_1, P_2, Q_2, R_1, R_2 are again seminormal.

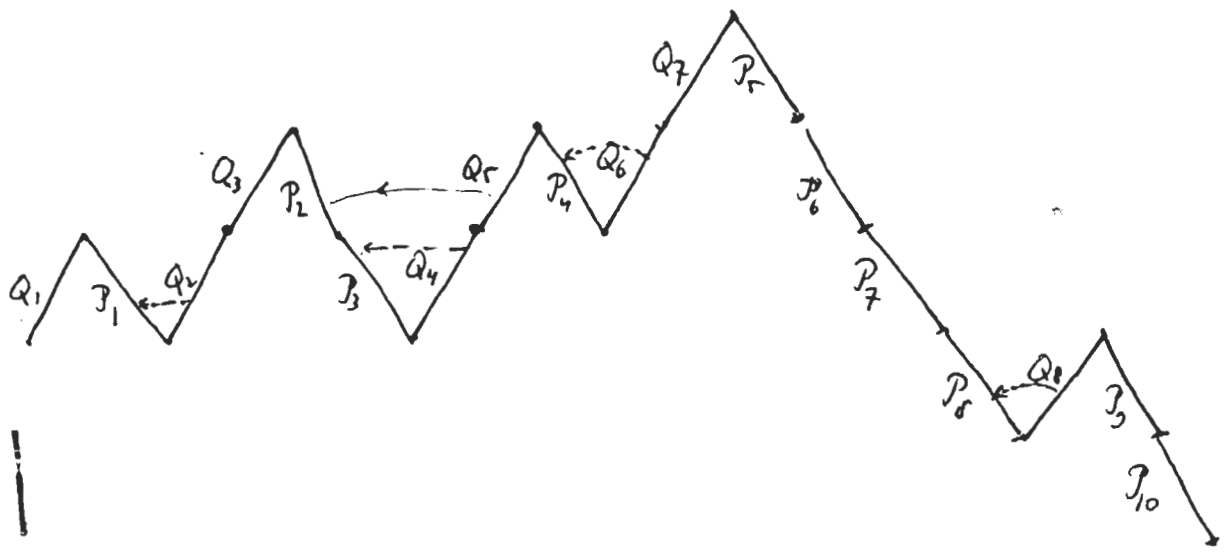
If we have introduced equivalence on the basis of β-reduction we can say that every expression is equivalent to a seminormal one. We illustrate this by an example:

$$\langle P_1\rangle\langle P_2\rangle\langle P_3\rangle\,[x:Q_1]\,[y:Q_2]\,\langle P_4\rangle\langle P_5\rangle[z:Q_3]\,x$$

We take the first AT-pair $\langle P_3\rangle[x:Q_1]$, and let both $\langle P_1\rangle$ and $\langle P_2\rangle$ jump over it. Next $\langle P_2\rangle$ forms

an AT-pair with $[y : Q_2]$, and we let $\langle P_1 \rangle$ jump over it. We see a third AT-pair $\langle P_5 \rangle [z : Q_3]$ and we let $\langle P_1 \rangle$ and $\langle P_4 \rangle$ jump over it. So we get $\langle P_3 \rangle [x : Q_1] \langle P_2 \rangle [y : Q_2] \langle P_5 \rangle [z : Q_3] \langle P_1 \rangle \langle P_4 \rangle$.

We note that the abstractors appear in the same order as in the original expression. Moreover, an abstractor becomes part of an AT-pair if in the original expression it had more applicators than abstractors in front of it; but the converse is not true: there are other cases as well. The following illustration may be helpful. In the original formula we have a sequence of $\langle P_i \rangle$'s and $[x_j : Q_j]$'s which we draw as descending and ascending lines like this:



If some Q looks to the left, its eye may fall on a P: with this P it will form an AT-pair (This happens with $P_1 Q_2$, $P_3 Q_4$, $P_2 Q_5$, $P_4 Q_6$, $P_8 Q_8$). The Q's that can look to infinity to the left, will stay alone. Furthermore, if we look at the figure from the right, the only P's which are visible are just the applicators that appear at the end in the seminormal form. Therefore, we can write down the seminormal form almost automatically:

$$Q_1 (P_1 Q_2)\ Q_3 (P_3 Q_4)\ (P_2 Q_5)\ (P_4 Q_6)\ Q_7\ (P_8 Q_8)\ P_5\ P_6\ P_7\ P_9\ P_{10}$$

(which has to be closed off at the end by a variable or by τ).

This introduction on seminormality was only meant for orientation. We shall give formal definitions later.

2. Motivation. It is the main idea of this note that it might be advantageous to restrict our $AT\lambda\tau$ - calculus to seminormal expressions, and to reformulate all operations in such a way that they become operators

in that sub-class. The author thinks that in this way we can circumvent a difficulty that we have in other treatments of the calculus. Let us explain this in terms of the presentation in [1]. The correctness of a tree is expressed in [1] by requiring a category check at every applicational knot. This check possibly involves a number of beta reductions based on other applicational knots. For theoretical purposes (and possibly also for avoiding duplication in correctness checking) we might like to check the applicational knots in such an order that the beta reductions we need for checking a knot, are based on applicational knots we checked already. The treatment in [1] does not seem to provide such an order, but the one with seminormal expressions is satisfactory in this respect.

Another advantage of seminormality is that it will be quite easy to keep track of the "external content" of the sub-expressions of an expression. We

use the term "extended context" as follows: in an expression

$$\mathcal{R}_1\ \mathcal{R}_2 \ldots\ \mathcal{R}_n\ \mathcal{B},$$

where each $\mathcal{R}_i$ is either an abstractor or an AT-pair, the sequence $\mathcal{R}_1 \mathcal{R}_2 \ldots \mathcal{R}_n$ is called the <u>extended context</u> of $\mathcal{B}$. (We use the word "extended" since in the AUTOMATH terminology the word "context" refers to a sequence of abstractors only). It is used here for a subexpression $\mathcal{B}$ at the end of the expression, but it can also be done for "internal" subexpressions: in

$$\mathcal{R}_1\ \mathcal{R}_2\ [\ x:\ \mathcal{R}_3 < \mathcal{R}_4\ \mathcal{B} > \mathcal{C}\ ]\ \mathcal{D}$$

the extended context of $\mathcal{B}$ is $\mathcal{R}_1\ \mathcal{R}_2\ \mathcal{R}_3\ \mathcal{R}_4$, and the one of $\mathcal{C}$ is $\mathcal{R}_1\ \mathcal{R}_2\ \mathcal{R}_3$.

Let us also comment on the disadvantages of a system working with seminormal expressions only. First, it takes some trouble to get to the seminormal form of an ordinary expression. And things like beta-reduction and the typing operation are slightly more complicated

than with the ordinary calculus. The difference is that the seminormalization operation has to be carried out, possibly quite frequently. Yet this operation is quite superficial: it just means interchanging a few sub-expressions.

As a further disadvantage it might be felt that the seminormalization throws information overboard which we might like to keep. If in

$$\ldots \; \langle P_1 \rangle \langle P_2 \rangle [x : Q] \ldots$$

we shift $\langle P_1 \rangle$ to the right of the AT-pair, we get

$$\ldots \; \langle P_2 \rangle [x : Q] \langle P_1 \rangle \ldots ,$$

and the information that P_1 does not depend on x, has been thrown overboard. If the question ever comes up we have to inspect P_1 again and find that it contains no x. If we fear this might become a waste of time we might decide to keep track of such non-dependencies by a kind of administration that is attached to our calculus without properly belonging to it.

Needless to say, in ordinary lambda calculi this kind of loss of information happens all the time, but in this case of seminormalization one might feel it is a self-inflicted wound.

3 Seminormalization. We define a mapping, to be denoted sn, of the set of all $AT\lambda\tau$-expressions onto the set of seminormal ones. It is defined recursively by

(i) $sn(\tau) = \tau$, $sn(x) = x$ (if x is a variable).

(ii) $sn([x:P]Q) = [x: sn(P)]\, sn(Q)$

(iii) $sn(\langle P\rangle Q) = \langle sn(P)\rangle sn(Q)$,

unless $sn(Q)$ has the form $\langle R\rangle [x:S]U$. In that case we define

$$sn(\langle P\rangle Q) = \langle R\rangle [x:S]\, sn(\langle P\rangle U).$$

We note that whenever some $\langle\ \rangle$ or some $[\ :\ ]$ appears for the first time (on the right) in the evaluation of sn, it is in the form $\langle sn(\ldots)\rangle$ or $[\ : sn(\ldots)]$, and therefore

in the exceptional case of (iii) the R and S can be written in the form $R = sn(R_1)$, $S = sn(S_1)$. This observation makes it easy to prove that

$$sn(sn(P)) = sn(P)$$

for all ATλτ-expressions P. The proof is by induction with respect to $l(P)$ (the length of the expression P).

We also note that

$$l(sn(P)) = l(P)$$

for all P. Assuming that P_0 is the shortest expression for which this is false, we easily get to a contradiction.

It is also possible to show that $sn(P)$ has the property described in the introduction: it nowhere contains $\langle\,\rangle\langle\,\rangle[:]$. However, this property of nowhere containing such a sequence is difficult to express and to handle, so we better decide to define the notion of "seminormal expression" by saying that an expression P is called seminormal if

$$sn(P) = P.$$

It can now be shown that any seminormal expression has the form

$$C_1 \dots C_n \quad \mathcal{D}_1 \dots \mathcal{D}_m \, \rho \qquad (3.1)$$

where

(i) each C_i is either of the form $[x_i : Q_i]$ where Q_i is seminormal, or of the form $\langle P_i \rangle [x_i : Q_i]$ where both P_i and Q_i are seminormal,

(ii) each $\mathcal{D}_j$ has the form $\langle R_j \rangle$, where R_j is seminormal,

(iii) ρ is either τ or a variable.

We allow that $n=0$ (i.e., the string $C_1 \dots C_n$ is empty) and also that $m=0$ (i.e., the string $\mathcal{D}_1 \dots \mathcal{D}_m$ is empty).

The string $C_1 \dots C_n$ is called an extended context. And we shall refer to the parts $C_1, \dots, C_n, \mathcal{D}_1, \dots, \mathcal{D}_m$ as the main-line constituents of the expression (3.1). They are obviously uniquely determined. The $x_1, \dots, x_n$ are called the main-line variables of $C_1 \dots C_n$.

The expression (3.1) will be called unburdened if $m=0$, and burdened if $m>0$.

4. Seminormal beta-reduction. In [1], we considered long distance β-reductions and shifts. We now prefer to take as basic steps long distance β-reduction and AT-removal.

AT-removal is simple: if some subexpression is $\langle P \rangle [x : Q] R$, and if R does not contain the variable x, then we reduce that subexpression to R.

In long distance β-reduction we have at some end-point of the tree a variable x, and that x is bound by the abstractor of an AT-pair $\langle P \rangle [x : Q$

Seminormal
~~The~~ long-distance β-reduction I can now be described in two steps

(i) replace that x at that endpoint by P.

(ii) seminormalize the resulting formula.

As to (ii) we remark that if we apply (i) to a seminormal formula, the seminormality may be lost.

A thing we are not going to treat in detail, but which will always be on the back of our minds, is

that seminormal beta-reductions possess ordinary beta equivalence: If P turns into Q by seminormal beta reduction, then P and Q are beta-equivalent in the sense of ordinary lambda calculus.

5. The operation sncat. Let P be a seminormal expression. We represent it in the form (3.1).

We are not defining the degree of P yet, but we already say that P is called "of degree 1" if $\rho = \tau$.

If P is not of degree 1, ρ is a variable, and, according to the way we have built our expressions, it is one of the x_i (in the notation in which (3.1) was explained in section 3). That is, this x_i occurs in an abstractor $[x_i : Q_i]$. We now define the sncat (P) by

$$\text{sncat}(C_1 \dots C_n \, \mathfrak{D}_1 \dots \mathfrak{D}_m \, x_i) = \text{sn}(C_1 \dots C_n \, \mathfrak{D}_1 \dots \mathfrak{D}_m \, Q_i).$$

We note that the evaluation of the right-hand side does not affect $C_1 \dots C_n$. And it does not affect

the $D_1, \ldots, D_m$ separately. It only affects the order of the sequence $D_1 \ldots D_m U_1 \ldots U_k$, where $U_1, \ldots, U_k$ are the main-line constituents of Q_i.

6. Correctness. We shall develop the notion of correct "extended context" and "correct expression" (all in seminormal form) by simultaneous recursion.

As abbreviations, we shall write $A \in CEC$ in order to express that A is a correct extended context and $B \in CE$ in order to say that B is a correct expression.

The rules (i), (ii), (iii) will take care of pronouncing new extended contexts; (iv) and (v) of pronouncing new unbundled expressions, and (vi) of pronouncing new bundled ones.

(i) The empty context is correct.

(ii) If $W \in CEC$, and $WQ \in CE$, and if x is different from the variables in W and Q, then

$$W[x:Q] \in CEC.$$

(iii) If $W \in CEC$, $WP \in CE$, $WQ \in CE$, (if WP is not of degree 1,) and if there are sequences of seminormal beta-reductions that reduce

$$\text{sncat}(WP) \quad \text{and} \quad WQ$$

to one and the same expression, then (with a new variable x)

$$W\langle P\rangle[x:Q] \in CEC.$$

(As usual, we have neglected the matter of renaming bound variables: we can take the point of view that indications of reference depths are the essential thing, and that it is just "syntactic sugaring" to give names to the variables).

(iv) If $W \in CEC$ then $W\tau \in CE$.

(v) If $W \in CEC$, and if x_i is one of the main-line variables of W, then $Wx_i \in CE$.

(vi) If $W \in CEC$, if x_i is one of the main-line variables of W, if $m > 0$ and if the seminormal

$\mathcal{P}$ has the form $W < R_1 > \dots < R_m > x_i$, when

$$sncat(\mathcal{P}) \in CE$$

then $\mathcal{P} \in CE$.

7. Consequences. As a consequence of these production rules we remark that a necessary condition for the correctness of a bundled expression $\mathcal{P}$ is that for some positive number k the k-th iterate $sncat^k(\mathcal{P})$ is unbundled.

In connection with $sncat^k$ we note that to every $\mathcal{P}$ (irrespective whether $\mathcal{P}$ is or is not correct), we can attach a positive integer $deg(\mathcal{P})$ (the degree of $\mathcal{P}$) such that $deg(\mathcal{P}) = 1$ for the $\mathcal{P}$'s we already defined to be of degree 1, and

$$deg(\mathcal{P}) = deg(sncat(\mathcal{P})) + 1$$

for all other $\mathcal{P}$.

By a careful analysis of the production rules it is not hard to show that

if $W_1 \in CEC$, $W_1 W_2 \in CEC$, $W_1 Q \in CE$ then $W_1 W_2 Q \in C$

Using this we can show the unbundled case of:

if $P \in CE$, $\deg(P) > 1$, then $\mathrm{sncat}(P) \in CE$, which is trivial if P is bounded.

In order to express further properties, we write $P \geq_1 Q$ if P turns into Q by a single (semi-normal) beta reduction step (AT-removal or long existence), and $P \geq Q$ if either $P = Q$ or P turns into Q by a finite sequence of single reductions ($P \geq_1 P_1 \geq_1 P_2 \geq \cdots \geq_1 P_n = Q$). And we write $P \downarrow Q$ if R exists with $P \geq R$, $Q \geq R$.

The essential things we have to prove are:

(i) If $P \in CE$, $P \geq_1 Q$ then $Q \in CE$.

(ii) If $P \in CE$, $P \geq_1 Q$ then $\mathrm{sncat}(P) \downarrow \mathrm{sncat}(Q)$

(iii) (Church-Rosser) If $P \in CE$, $P \geq Q$, $P \geq R$ then $Q \downarrow R$

(iv) (Strong normalization) If $P \in CE$ then there exists an integer N such that for every reduction sequence

$$P \geq_1 P_1 \geq_1 P_2 \geq_1 \cdots \geq_1 P_n$$

we have $n \leq N$.

According to the general theory in [2', 3] there is

little doubt as to the correctness of (i) to (iv), in spite of the fact that we have a slightly different notion of beta reduction. But the interesting question is whether we can give proofs for the seminormal cases of (i)-(iv) which are simpler than the proofs for the general case.

(8.) Remarks on correctness checking. The production rules of section 5 immediately suggest an algorithm for checking the correctness of a given expression. The expression P is either $W\tau$ or $W x_i$ or $W\langle R_1\rangle \ldots \langle R_m\rangle \rho$, and we first check the correctness of the extended context W. The cases $W\tau$ and $W x_i$ are trivial. In the remaining (burdened) case we follow the sequence $\mathrm{sncat}(P), \ldots, \mathrm{sncat}^{m-1}(P)$, (where $m = \deg(P)$) until we get an unburdened expression. If they are all burdened, then P is incorrect (in particular if $m > 0$ and $\rho = \tau$, then P is incorrect). If $\mathrm{sncat}^k(P)$ is the first unburdened one, we check that one according to the rules (i)-(iv).

It is only in this last expression $\mathrm{encat}^k(P)$ that we find a "mate" (i.e. an abstractor that helps to form an AT-pair) for $\langle R_1 \rangle$. At that moment we may have some abstractors $C_{h+1}, \ldots, C_m$ at the end, which play no role in the mating:

$$\mathrm{encat}^k(P) = C_1 \ldots C_h \, C_{h+1} \ldots C_m \, e,$$

where C_h is an AT-pair, $C_1, \ldots, C_{h-1}$ are either abstractors or AT-pairs, and $C_{h+1}, \ldots, C_m$ are abstractors. In that case we need not check the correctness of the extended context beyond C_h: the correctness of $C_{h+1}, \ldots, C_m$ is guaranteed by virtue of where they came from.

We mention a further case where we can make a shortcut in the correctness check. In case (iii) of section 5 it may happen that $\mathrm{encat}(WP)$ is just identical to WQ. Then we need not check $WQ \in CE$ since it is a consequence of $WP \in CE$ (see section 6).

9.) Addendum. We add some further "desirable" properties to those listed in section 6. Some of these might play a role in the proofs of those properties in section 6.

(v) If $\frac{W \in CEC}{WQ \in CE}$, $WV \in CE$, $WV \geq WV_1$, and $sn(W < Q > V_1) \in CE$ then $sn(W < Q > V) \in CE$.

(vi) If $W[x : T_1]V_1 \downarrow W[x : T_2]V_2$ then $WT_1 \downarrow WT_2$.

(vii) If $W \in CEC$, $WR \in CE$, $WQ \in CE$, $sncat(WR) = Wy$, $W < Q > R \in CE$ then

$$sncat(W < Q > R) = sn(W < Q > y)$$

10.) An alternative definition. The definition of correctness in section 6 depended on the operation sncat as defined in section 5. Instead of this, we can use a notion sncat2 we shall explain here.

The notion of sncat2 will be defined recursively on a class of semi-normal expressions we have to define at the same

time. That class will not contain any expression of degree 1, so the domain of sncat2 will be a subset of the domain of sncat. We shall introduce a symbol ∞ for "undefined", and describe sncat2(P) for all (seminormal) $AT\lambda\tau$-expressions P, and later we say that the domain of sncat2 consists of those P for which sncat2 $\neq \infty$. We now explain sncat2 recursively as follows. According to (3.1) we write $P = C_1 \dots C_n D_1 \dots D_m e$. If $e = \tau$ we take sncat2(P) $= \infty$. If $e = x_i$, for some i with $1 \leq i \leq n$, we consider two cases:

(i) C_i has the form $[x_i : Q_i]$. We now take (just like in the definition of sncat in section 5)

$$\text{sncat2}(P) = \text{sn}(C_1 \dots C_n D_1 \dots D_m Q_i).$$

(ii) C_i has the form $\langle P_i \rangle [x_i : Q_i]$. Then we take

$$\text{sncat2}(P) = \text{sn}(C_1 \dots C_n D_1 \dots D_m R),$$

where R is such that $\text{sncat2}(C_1 \dots C_{i-1} P_i) = C_1 \dots C_{i-1} R$.

We note that the recursion works because of the fact that $C_1 \dots C_{i-1} P_i$ is shorter than $C_1 \dots C_n D_1 \dots D_m Q_i$).

In some stages of the theory $\mathcal{F}_{\mathrm{cat}2}$ seems to be more useful than $\mathcal{F}_{\mathrm{cat}}$ itself, but in some other stages it is not. Only after having the complete theory for both cases, we will be in a position to say which one is to be preferred.

References

1. N.G. de Bruijn. A new definition of correctness of expressions in λ-typed λ-calculus. Report M 12, (May 1983).

2. D.T. van Daalen. The language theory of Automath. Ph.D. Thesis Eindhoven University of Technology, 1980.

3. R.P. Nederpelt. Strong normalization in a typed lambda calculus with lambda structured types. Ph.D. Thesis, Eindhoven University of Technology, 1973.