EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics

Memorandum 1980-01

January 1980

An approach to theorem proving on the basis of a typed lambda-calculus

by

R.P. Nederpelt

University of Technology
Department of Mathematics
PO Box 513, Eindhoven
The Netherlands.

An approach to theorem proving on the basis of a typed lambda-calculus

R.P. Nederpelt (*)

Abstract and introduction

This paper describes a system of typed lambda-calculus suited to representing mathematical texts, and discusses some theorem proving aspects of the system. In part 1 a formal exposé is given of the system, with comments on the principles chosen. A natural manner of rendering mathematical texts in the system will be explained in part 2. Finally, in part 3 the system will be investigated as regards its potentials for (partial) theorem proving.

An idea for a "completely" formalized, yet natural language for expressing mathematical texts was conceived by N.G. de Bruijn. In 1968 he developed the mathematical language Automath. Automath has since been extensively applied and tested on numerous mathematical topics, language theory has been developed for establishing its computational soundness, and a computer programme has been produced for checking the formalized texts. For Automath and applications, cf. references [1], [2], [8] and [5].

Automath is essentially a typed lambda-calculus presented in a modified form in order to make it more accessible to the customer. The underlying typed lambda-calculus, which we call Λ, is relatively simple and in a sense natural. In contrast to types in the usual typed systems, types in Λ behave like terms, and do not look different to terms. The system Λ obeys the same "nice properties" as does Automath, such as the Church-Rosser property, unique (strong) normalization and closure (see section 1.6). For language theory of Automath and Λ, see [6], [3] and [4].

Since Λ has a simple, transparent structure, it would seem promising to investigate its practical theorem proving aspects. However, there has not been much experience in this direction. Some considerations and comments, arising from a Λ-text example presented in this paper, may nevertheless be helpful for practical applications to theorem proving.

(*) Thanks are due to A.V. Zimmermann for remarks concerning the use of the English language.

1. A concise definition of the system Λ

1.1. Notations

Like Automath, Λ uses notations that deviate from lambda-calculus conventions. For $\lambda_x B$, x having type A, we write $(A\lambda_x)B$. Instead of (BA), i.e. "function" B applied to "argument" A, we write (Aδ)B. Here δ is the signal for a functional application. There are two main reasons for inverting the order of function and argument: (1) there is an analogy between the dual operations "abstraction" (leading from B to $(A\lambda_x)B$) and "application" (leading from B to (Aδ)B); (2) the inversion is very practical when rendering mathematical texts by means of Λ (see e.g. sections 2.6 and 2.7).

1.2. Terms

We introduce the set T^o of (open) *terms* by the following recursive definition; the alphabet consists of variables, brackets and the symbols λ and δ.

(1) The empty term is a term of T^o; each variable is a term of T^o.

(2) If A and B $\in T^o$ and if x is a variable, then $(A\lambda_x)B$ and (Aδ)B are terms of T^o.

The empty term is rendered invisibly; examples of other terms are: x; $((\delta)\lambda_x)y$.

The set T^c is defined as the closed fragment of T^o, i.e. the set of all closed terms present in T^o; a *closed* term is a term without free variables. We say that G is a *subterm* of F, if F and G are terms and G occurs in F. If, moreover, the first symbol occurring after G in F (if any) is δ or λ, we say that G is a *genuine* subterm of F. (This definition implies that z and $(\lambda_y)x$ are genuine subterms of $((\lambda_x)(\lambda_y)x\delta)z$, but (λ_y) is a subterm that is not genuine.)

1.3. Typing

We need types to enable us to attach "classes" to "objects": if C is a term representing an object, then the type of C represents the class of that object. In this context we note that the empty term acts as a kind of class for all classes. See also section 2.2.

Formally, we define types as follows. If $(B\lambda_x)$ is a subterm of term A, and x occurs as a variable in A that is bound by the λ mentioned, then we say

that the type of the bound x, relative to A, is B. In the case that C is a subterm of $A \bullet T^O$ and C ends in a variable x (so $C \equiv Dx$ for some term D), where x is bound in A, then the type of C, relative to A, is the concatenation of D and the type of x. We denote the type of C, relative to A, by $Typ_A[C]$, or, when no confusion is possible: Typ[C].

Example: $Typ[((\delta)y\lambda_x)((x\delta)z)x] \equiv ((\delta)y\lambda_x)((x\delta)z)(\delta)y$.

We note that the types are not given beforehand as a separate set. They can be calculated for some subterms of a term in T^O. In particular, all genuine subterms of a closed term A that end in a variable, have a calculable type. So for $A \in T^O$, Typ_A is a partial function from the set of all subterms of A to T^O. The iterates of Typ are denoted, as usual, by Typ^2, Typ^3, etc.

1.4. Reductions

We provide T^O with the usual reduction relations, called α-, β- and η-reduction. Since α-reduction is a mere renaming of variables, we prefer to consider T^O as being the set of all α-equivalence classes, terms being α-equivalent if and only if one term reduces to the other by means of α-reduction.

The essential reduction for T^O is *β-reduction* (symbol: $>_\beta$), which is the formalization of the application of a function to an argument. It is induced by the rule: $(A\delta)(B\lambda_x)C >_\beta {}^x_A C$, the latter term being the result of (simultaneously) substituting A for all x's free in C. We also have *η-reduction*, induced by: $(A\lambda_x)(x\delta)C >_\eta C$ if x does not occur freely in C.

As usual, *conversion* is the equivalence relation generated by $>_\beta$ and $>_\eta$. We denote conversion by the symbol ~.

1.5. Strong functionality

In systems of natural reasoning there is a natural desire to restrict the functional applicability. In words: $(A\delta)B$ is only permitted as a subterm of a given F, if B has a domain, say C, and if A fits in domain C. We shall express "having a domain" and "fitting in" by means of the function Typ.

Formally, we define: $F \in T^C$ is *strongly functional* if and only if the following condition applies: for all genuine subterms of the form $(A\delta)B$ it holds that there exist a non-negative number n, a variable y and terms C and D such that

(i) $Typ^n[B]$ exists and $Typ^n[B] \sim (C\lambda_y)D$,

(ii) Typ A exists and Typ A ~ C.

We define Λ as the set of all closed terms that are strongly functional.

Note: For theoretical purposes, a much weaker form of functionality is sufficient to ensure the validity of the "nice properties" (see the next section). One may define "weakly functional" terms, which have a functional structure comparable to that in usual typed lambda-calculuses. The weak system is helpful, proofs of (strong) normalization for Λ being given with the aid of the analogous proofs for the weaker system, which are relatively easy. A definition of weak functionality is given in an appendix to this paper; for details, see [4] and [6].

1.6. Properties of Λ

(1) The *Church-Rosser* (or *diamond*) *property*, i.e.: if $A \in \Lambda$ reduces to B, and A reduces to C, then B and C have a common reduct.

(2) *Closure*, i.e.: if $F \in \Lambda$, then $Typ[F] \in \Lambda$; moreover, if F reduces to G, then $G \in \Lambda$.

(3) (Unique) *normalization*, i.e.: if $F \in \Lambda$, then there is a unique normal term H such that F reduces to H. (H is *normal* when there is no G such that $H >_\beta G$ or $H >_\eta G$.)

(4) *Strong normalization*, i.e.: each reduction sequence starting from a strongly functional term, terminates.

For proofs of these theorems, see [6] and [4].

2. Expressing mathematics in Λ

2.1. Translation of a text

There is a standard way, in a sense natural, for translating mathematical texts into Λ. We shall comment on the principles of this manner of translating. The idea is, that a mathematical text transforms into a long-drawn term of Λ. Not only mathematical entities such as sets and functions, present in the original text, become subterms of this term, but also text units such as theorems and assumptions have their direct counterparts in subterms. The order of the text units in the original reasoning is generally maintained in the translation.

For obtaining a term of Λ, having no free variables, one should in principle have a text that is complete in a double meaning: the text should not have gaps in the reasoning or argumentation, and all foreknowledge (axioms, theorems, definitions used in the text) must be explicitly given. In practice one only translates a portion of text when all foreknowledge is accessible in translated form, so that the text under consideration becomes, after translation, a mere extension of an already existing (possibly very long) Λ-term.

We shall now discuss a possible way of translating some mathematical notions or text units.

2.2. Sets and propositions

Our (long) Λ-term opens with two subterms: (λ_τ) and (λ_π). We think of τ as being the class of all sets, and π as being the class of all propositions.

If we wish to express, somewhere in the translation, that variable s must denote a set, we write $(\tau\lambda_s)$ in our Λ-term. Then $\mathrm{Typ}[s] \equiv \tau$, in correspondence with our interpretation of τ and of Typ. Analogously, if we wish to regard variable p as a proposition, we write $(\pi\lambda_p)$.

An element x of set s may now be introduced by embodying the subterm $(s\lambda_x)$ in our term. For the analogous subterm $(p\lambda_t)$, where p is a proposition, there is a nice and practical interpretation: t is a *proof* of p. (This so-called *propositions-as-types notion* has fairly recently been introduced by several investigators, among others De Bruijn; for comment, see [4]).

In this manner one obtains interpretations for four different grades of terms. The 0-grade only contains the empty term, to be interpreted as the class of all classes. The 1-grade contains τ, the class of all sets, and π,

the class of all propositions. The sets and the propositions themselves can be found in the 2-grade. Finally, the 3-grade contains elements of sets and proofs of propositions.

Hence, if X is an element (or proof) in the 3-grade, then Typ[X] is a set (or proposition) in the 2-grade, $Typ^2[X]$ is τ (or π respectively) and $Typ^3[X]$ is the empty term. It is striking that we only need these four grades for representing a large section of mathematics, although Λ has possibilities for arbitrary n-grades (n being a non-negative number).

2.3. Functions

It is convenient to use the functional structure of lambda-calculus in describing functions. For example, the identity function on A can obtain the term $(A\lambda_x)x$ as its counterpart in Λ. We take the term $(A\lambda_x)A$ as type of this function, usually written A^A. This interpretation of $(A\lambda_x)A$ is not self-evident, but such a *type-valued function* is, again, very practical in use. We note that this policy corresponds with the formal identity $Typ(A\lambda_x)x \equiv (A\lambda_x)A$. For further explanation, see [4].

Following the above convention concerning type-valued functions, there is a plausible interpretation for the term $(p\lambda_t)q$, where p and q are propositions, viz.: $p \Rightarrow q$. This can be understood as follows. If u is a proof of q (so $Typ[u] \equiv q$ according to the propositions-as-types notion), then function $(p\lambda_t)u$ conveys any proof t of p into proof u of q. Hence $(p\lambda_t)u$ proves the implication $p \Rightarrow q$, so that the type of $(p\lambda_t)u$ must be $p \Rightarrow q$. But $Typ[(p\lambda_t)u]$ is $(p\lambda_t)q$, so the latter represents the implication.

Analogously, term $(A\lambda_x)q$, where A embodies a set and q a proposition, represents: $\forall_{x\in A}[q]$. Here q is a term that may contain the free variable x.

2.4. Assumptions and introductions

The text unit "Let $x \in A$" introduces a variable x of type A. In translation this becomes $(A\lambda_x)$. Analogously, the assumption "Assume p" can be translated by $(p\lambda_t)$. Note that the latter mode of translation is in accordance with the propositions-as-types notion: the subterm $(p\lambda_t)$ can be read as: "Let t be a proof of proposition p".

2.5. Axioms, axiomatic notions

Axioms and axiomatic notions may be regarded as introductions (or assumptions) with an unbound validity range. For example, the primitive notion "natural number" can be introduced by means of the subterm $(\tau\lambda_{\mathbb{N}})$. The first Peano axiom, "1 is a natural number", reads: $(\mathbb{N}\lambda_{one})$, and so on.

2.6. Definitions

When object α of class β is abbreviated by variable x, then it is to be understood that each occurrence of x "means" α. This is essentially what the definition $x := \alpha$ does. Let A and B be translations into Λ of α and β. Then we can write the definition in translation as $(A\delta)(B\lambda_x)$, since β-reduction enables us to again replace by A every x bound by this λ. Moreover, by strong functionality both A and x must have type (convertible to) B. These observations imply that the effect of the insertion of $(A\delta)(B\lambda_x)$ is that x "means" A.

2.7. Theorems, lemmas and intermediate results

In translating theorems, we lean heavily on the propositions-as-types notion. Let B be the translation of a proposition that we regard as a theorem, and let A be the translation of its proof. Then we may insert the subterm $(A\delta)(B\lambda_t)$, expressing both the theorem and its proof. By strong functionality, $Typ[A] \sim B$, in accordance with "A proves B". Variable t may be regarded as a name of the proof A. Theorem B may later be applied by referring to its proof, which can be done by calling the name t of the proof.

Lemmas and intermediate results may be treated analogously.

2.8. Deduction rules and logic

We shall briefly comment on the way in which logic can be incorporated. By introducing an axiomatic notion "contradiction": $(\pi\lambda_{cd})$, we can express the negation $\neg$ p of proposition p as $p \rightarrow cd$, or, in translation: $(p\lambda_t)cd$. The logical connectives $\wedge$, $\vee$ etc. now can be expressed by means of the implication and the negation.

The universal quantifier is already "present" in Λ, as we saw in 2.3. The existential quantifier $\exists$ then can be easily expressed as $\neg \forall \neg$.

The elimination and introduction rules of natural deduction now are implicitly present in the system. They are a result of the natural language structure, and need not be introduced as primitive rules or axioms. See also [7].

When wishing to apply classical logic, one adds the double negation rule: $\neg\neg\ p \Rightarrow p$ as an axiom.

2.9. Remarks on some translation difficulties

There are a number of pecularities that hamper the translation of a mathematical text into Λ. We mention a few. (For more extensive comments on these topics, see [5] and [4].)

(1) The system Λ has "uniqueness of types". That is to say: if A converts to B, then Typ[A] converts to Typ[B]. This presents practical difficulties as to the hierarchy of types. For example, if x is a natural number, then x is not automatically a real number as well, since $\mathbb{N}$ and $\mathbb{R}$ are obviously non-convertible. A way out is to write in Λ a mechanism of embedding and "exbedding", to enable us to deal with sets and subsets.

(2) Two proofs of a certain statement are in principle different. This gives undesirable effects in the case in which only the existence of a proof matters, not its nature. For example, the natural logarithm ℓn will have two arguments in Λ: a number x, and a proof s that this number is positive. So in fact we should not write $\ell n\ x$, but $\ell n(x,s)$. If s and t are two different proofs of the positiveness of x, however, then nevertheless $\ell n(x,s)$ and $\ell n(x,t)$ should be "equal". One can write in Λ an axiom yielding such an "irrelevance of proofs" in these cases.

(3) In Λ there is no primitive equality, apart from conversion. So some forms of equality (e.g. between sets, and between numbers) have to be expressed axiomatically. This treatment of equality is in principle feasible, but in practice somewhat cumbersome.

(4) When Λ is used in the form as described above, it gives rise to numerous repetitions inside the Λ-term. See the example in section 3.2. Front parts of subterms are often repeated; they are subterms themselves, but since they end in the empty term, they cannot be abbreviated as is done with definitions (cf. section 2.6). It is not hard, however, to extend Λ in such a manner that the abbreviations meant can be carried out.

3. An approach to theorem proving on the basis of Λ

3.1. The shape of a translated mathematical text

When following the translation conventions discussed in section 2, one obtains a single Λ-term that may be considered a concatenation of *fragments*. Each fragment is a subterm ending in the empty term. There are three kinds of fragments:

1. the *initial* fragments, which stand at the heading of the term, namely (λ_τ) and (λ_π),
2. *primitive* fragments of the form $(A\lambda_p)$, A being a 1- or 2-grade term (see section 2.2),
3. *stating* fragments of the form $(A\delta)(B\lambda_x)$, B being a 1- or 2-grade term.

The role of the initial fragments will be clear. The primitive fragments are the translations of axioms and axiomatic notions. The stating fragments are translations of theorems, lemmas, intermediate results, but also of definitions.

3.2. Example of a text in Λ

As an example we render the first few lines of Jutting's complete translation of Landau's "Grundlagen" (see [5]). Jutting's translation is in Automath; we give the Λ-version. For reasons of economy (cf. section 2.9, note (4)) we draw a line when a repetition is meant. E.g., the fourth line in the subjoined Λ-text should read $((\pi\lambda_a)(\pi\lambda_b)\pi\lambda_{imp})$. Numbers 1 to 11 are extra-textual, only meant for numbering the fragments. The Λ-text below, read uninterruptedly, yields a single Λ-term. The content of each fragment will be explained afterwards.

1. (λ_τ)
2. (λ_π)
3. $((\pi\lambda_a)(\pi\lambda_b)(a\lambda_x)b\delta)$
   $(\text{———————}\pi\lambda_{imp})$
4. $((\pi\lambda_a)(\pi\lambda_b)(\pi\lambda_c)((b\delta)(a\delta)\text{imp }\lambda_i)((c\delta)(b\delta)\text{imp }\lambda_j)(a\lambda_x)((x\delta)i\delta)j\delta)$
   $(\text{———————————————————————————————}(c\delta)(a\delta)\text{imp }\lambda_{trimp})$
5. $(\pi\lambda_{cd})$
6. $((\pi\lambda_a)(cd\delta)(a\delta)\text{imp }\delta)$
   $(\text{———}\pi\lambda_{not})$

7. $((\pi\lambda_a)((a\delta)\text{not }\delta)\text{not }\delta)$

   $(\text{———}\pi\lambda_{nn})$

8. $((\pi\lambda_a)(a\lambda_{a1})((a\delta)\text{not }\lambda_x)(a_1\delta)x\delta)$

   $(\text{————————}(a\delta)nn\lambda_{nni})$

9. $((\pi\lambda_a)((a\delta)nn\lambda_w)a\lambda_{dn})$

10. $((\pi\lambda_a)(cd\lambda_{c1})(((a\delta)\text{not }\lambda_x)c1\delta)(a\delta)dn\delta)$

    $(\text{————————}\ a\lambda_{dne})$

11. $((\pi\lambda_a)(\pi\lambda_b)((a\delta)\text{not }\lambda_n)((cd\lambda_x)(x\delta)b\delta)dne\delta)(n\delta)(b\delta)(cd\delta)(a\delta)\text{trimp }\delta)$

    $(\text{————————————————————————————}(b\delta)(a\delta)\text{imp }\lambda_{th})$

Fragments 1 and 2 are initial fragments, 5 and 10 are primitive fragments. The others are stating fragments, where 3, 6 and 7 concern definitions; 4, 8, 9 and 11 may be regarded as theorems.

The content of the fragments is the following. Fragments 1 and 2 need no comment. In 3 the implication is defined (see also section 2.3), 4 states the transitivity of implication. In 5 contradiction is introduced as an axiomatic notion. Fragment 6 defines negation (see also section 2.8), 7 the double negation. In 8 the theorem is proved that $a \Rightarrow \neg\neg a$ holds; 9 states, as an axiom, the double negation rule. Fragment 10 proves the falsum-principle. In 11, finally, the logical theorem $\neg a \Rightarrow (a \Rightarrow b)$ is proved.

3.3. The construction of a proof

We ignore the proof given in the first part of fragment 11 (previous section), and try to construct a proof independently. In this construction we follow a strictly formal approach; we do not appeal to any mathematical insight. To begin with, we transform the text above into normal form. This is, of course, a crude and inefficient thing to do, especially for longer Λ-texts, but we obtain so doing a clearer view on the principles of proving.

Fragments 1 to 10 transform into the normal Λ-term:
$(\lambda_\tau)(\lambda_\pi)(\pi\lambda_{cd})((\pi\lambda_a)(((a\lambda_y)cd\lambda_x)cd\lambda_w)a\lambda_{dn})$, consisting of four primitive fragments. Now a proof of the theorem has to be a term with type (converting to) Y, where $Y \equiv (\pi\lambda_a)(\pi\lambda_b)((a\lambda_x)cd\lambda_n)(a\lambda_{x0})b$, which is the normal form obtained from the term expressing the theorem:
$(\pi\lambda_a)(\pi\lambda_b)((a\delta)\text{not }\lambda_n)(b\delta)(a\delta)\text{imp}$.

We can describe the actual *proving state* as follows: on one hand we have a *stock* of variables and matching types, on the other hand there is a *target*, determined by one or more types. In our case, we have the following initial proving state: the stock consists of the "leading variables" of the four primitive fragments, namely τ, π, cd and dn, together with their types; the target is to find a term with type (converting to) Y.

In view of the shape of Y, it is appropriate to change the proving state: add a, b, n and x0 to the stock, with types as above, and change the target into a term X with type (converting to) b. None of the variables in the stock has a type that is b, converts to b, or ends in b. As to all stock-variables except dn, there is no way of changing the final variable of their types into b by reductions. Only dn can give us hope: the final a of its type is "internally bound", i.e.: bound by a λ that occurs inside the same fragment; so variable a can possibly be changed into b.

Hence we now direct our searching attempts at dn and, again,we change the proving state. We look for terms X_1 and X_2 (the new targets) such that $(X_2\delta)(X_1\delta)$dn has a type (converting to) b. Then X_1 must have type π and X_2 must have type $((X_1\lambda_y)cd\lambda_x)cd$, according to strong functionality. Now $Typ[(X_2\delta)(X_1\delta)dn] \equiv X_1$, as can be easily computed, so X_1 must be b. The remaining target is an X_2 with type (converting to) $Y_2 \equiv ((b\lambda_y)cd\lambda_x)cd$.

In view of the shape of Y_2, we add x to the stock of variables, with its type: $(b\lambda_y)$cd. Now the target becomes a term X_{21} with type cd. The only possibilities are to use dn, n or x from the stock. We choose to use n and have to find an X_{211} such that $(X_{211}\delta)n$ has type cd. Then X_{211} must have type a, and, indeed, X_{211} := x0 does the job.

Thus we have reached the final proving state, in which no target is left. Recapitulating: we found $X_{211} \equiv a$; $X_{21} \equiv (X_{211}\delta)n \equiv (x0\delta)n$; $X_2 \equiv ((b\lambda_y)cd\lambda_x)X_{21}$; $X_1 \equiv b$ and $X \equiv (X_2\delta)(X_1\delta)dn$. The requested proof is $(\pi\lambda_a)(\pi\lambda_b)((a\lambda_x)cd\lambda_n)(a\lambda_{x0})X$. Inspection shows that we have found the "same" proof as given in fragment 11 of the example in section 3.2 when written in *normal form*.

There were only a few choices to be made in this simple proving problem. Yet, if we had chosen to use dn instead of n, when looking for X_{21}, we would have returned to a prior proving state. Hence, in principle we could have been caught in a loop.

3.4. Remarks on partial theorem proving on the basis of Λ

A general strategy for theorem proving on the basis of normal forms can easily be derived from the construction example above. This *normal strategy* does indeed work in uncomplicated cases, but it fails when major mathematical tools are needed, such as induction. In such cases there is, presumably, only hope for a mechanical theorem prover when it is built in an interactive way: the theorem prover must be able to react to hints from the human textwriter.

For general use one has to abandon the transformation into normal form. The normal proving strategy, as explained above, can however be adapted for non-normal Λ-texts, such as given in section 3.2. The strategy itself then becomes more complicated. It is, for instance, not sufficient to regard the final variable of a certain "type-term", but one also has to consider all variables that can possibly replace this variable when reductions are applied. This leads to the tracing of certain *chains* of variables. The comparison of variables, being a major activity in the normal strategy, then has to be replaced by a method of comparing variable chains.

Summarizing, there appear to be possiblities for *partial* theorem proving on the basis of Λ, in particular when small gaps have to be bridged. For exacting proofs, however, a form of interaction between man and machine appears indispensable.

References

[1] *N.G. de Bruijn*, The mathematical language AUTOMATH, its usage and some of its extensions, Symposium on Automatic Demonstration, IRIA, Versailles, France, 1968. (Lecture notes in Mathematics, Vol. 125, Springer-Verlag, pp. 29-61, 1970.)

[2] *N.G. de Bruijn*, AUTOMATH, a language for mathematics, Lecture Notes prepared by B. Fawcett. Les Presses de l'Université de Montreal, Canada, 1973.

[3] *D.T. van Daalen*, A description of AUTOMATH and some aspects of its language theory. Proceedings of the Symposium APLASM. Vol. I, ed. P. Braffort, Orsay, France, 1973.

[4] *D.T. van Daalen*, The language theory of Automath, thesis, Technol. University Eindhoven, the Netherlands, 1980.

[5] *L.S. van Benthem Jutting*, Checking Landau's "Grundlagen" in the AUTOMATH system, thesis, Technol. University Eindhoven, 1977. (Mathematical Centre Tracts 83, Amsterdam, the Netherlands, 1979.)

[6] *R.P. Nederpelt*, Strong normalization in a typed lambda calculus with lambda structured types, thesis, Technol. University Eindhoven, the Netherlands, 1973.

[7] *R.P. Nederpelt*, Presentation of natural deduction, Recueil des Travaux de l'Institut Mathématique, Nouvelle série, tome 2 (10), p. 115-126, Symposium: Set Theory, Foundations of Mathematics, Beograd, Jugo-Slavia, 1977.

[8] *J. Zucker*, Formalization of classical mathematics in AUTOMATH, Actes of the International Logic Colloquium, Clermont-Ferrand, France, 1975.

Appendix

Weak functionality

We define the *norm* (the "functional skeleton") $\|A\|$ of a term A by the following partial definition:

(i) the norm of the empty term is the empty term; if x is a variable and $\|\mathrm{Typ}[x]\|$ exists, then $\|x\| := \|\mathrm{Typ}[x]\|$;

(ii) if A and B are terms for which $\|A\|$ and $\|B\|$ exist, then

$\|(A\lambda_x)B\| := (\|A\|\lambda_x)\,\|B\|$;

if, moreover $\|B\| \equiv (\|A\|\lambda_x)D$, then $\|(A\delta)B\| := D$.

If $\|A\|$ exists for some term A, then $\|A\|$ is a term as well, without free variables. There are no δ's in $\|A\|$. The essential step in norm calculation is the cancellation of adjacent pairs $(B\delta)(C\lambda_x)$ when some weak functional condition is obeyed as to "argument" B and "domain" C. This condition is expressed in the second part of (ii). The calculation of the norm of A breaks off prematurely (and A has no norm), or terminates in a unique norm $\|A\|$.

We say that a closed term F is *weakly functional* if $\|F\|$ exists. It is not hard to prove that strongly functional terms are weakly functional as well.