Зборник радова Математичког института, Нова серија, књ. 2 (10), 1977.
Recueil des travaux de l'Institut Mathématique, Nouvelle série, tome 2 (10), 1977,

Symposium: Set Theory. Foundations of Mathematics, Beograd, 29. 08. — 2. 09. 1977.

PRESENTATION OF NATURAL DEDUCTION

R. P. NEDERPELT*

Introduction

The merits of a system of natural deduction are not only determined by its value as a logical system in itself. Since it formalizes deductions in a manner close to intuitive reasoning, natural deduction can also be used as a (logical) framework for mathematical argumentation. One may say that many mathematical texts are tacitly based on a form of natural deduction, as regards the logical part of the deductive patterns.

Jáskowski and Gentzen constructed the first systems of natural deduction in the early thirties (see Prawitz [7, appendix C]). Many suggestions have been made since with a view to formalizing the natural deduction structure present in usual mathematical reasonings.

Text-books concerned with logic on this basis are, for instance, Quine [9], Suppes [10] and Kalish-Montague [5]. The incorporation of a natural deduction system in the common mathematical practice can be very useful, in particular for didactical purposes.

In section I of this paper we shall propose another system of natural deduction, resembling that of Kalish and Montague, which can be used for the logical part of mathematics. The system to be described is quite satisfactory in practice, as became apparent when applying it to undergraduate mathematics tuition.

A natural way of reasoning in mathematics has, however, more aspects than the logical ones. These other, non-logical aspects were isolated by N.G. de Bruijn. His investigations led to a system called "the mathematical language Automath" (see [1]), which may serve as a formal notational system for rendering mathematics in a natural manner. The system is founded on typed lambda-calculus, not on axiomatic set theory.

---

* The author is employed in the Mathematics department of the Eindhoven University of Technology in the Netherlands. —

Thanks are due to N.G. de Bruijn, D.T. van Daalen and R.C. de Vrijer for helpful comments, and to A.V. Zimmermann for remarks concerning the English language.

In section II of this paper we shall describe, in coherence with section I, the major principles of such a "system of natural reasoning". The description will be rather informal and incomplete.

It will also be shown how the rules of natural deduction can be expressed within the system, so that an important part of natural reasoning finds a formalized counterpart. In systems like this, a large part of everyday mathematics can actually be expressed, as is shown in e.g. Jutting [4] and Zucker [11].

The structure underlying a system like de Bruijn's can be made clearer by uniformations, leading to a system which is a typed lambda-calculus, the types themselves having a lambda structure. This uniform system will be described in section III of this paper. It does not have the natural aspects of the other systems. It has, however, a relatively simple and transparent structure and is therefore very useful for theoretical investigations into "systems of natural reasoning", e.g. with respect to (strong) normalizability. We shall give a precise description of this system and summarize some of its properties.

I. A practical system of natural deduction

With the aim of obtaining a practical system for natural deduction, directly applicable in everyday mathematics, we reformulate the introduction and elimination rules for $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$ and $\exists$ (see e.g. Prawitz [7] or [8]), with modifications to be described below.

Basic units in the systems we shall call *sentences*, written in a sequential (not a tree-like) order, one sentence below the other. A sentence can express something like an axiom, a theorem, a definition, an assumption or a derived statement. If desired, one may add comments, e.g. containing justification for a derived statement. Such justifications may be based on logical rules (like the introduction and elimination rules), on premisses, valid assumptions and previous results, but also on mathematical arguments; this part of the reasoning is not formalized in the present system.

As primitive symbols we have the logical constants $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$, $\exists$ and contradiction. We do not consider the logical constant $\Leftrightarrow$ primitive; it can be defined in the usual manner in terms of $\wedge$ and $\Rightarrow$.

We note that in mathematical practice the following observation is often used: if "F implies G" is a derived rule, then a proof of F suffices as a proof of G. (Thus a proof of b is also a proof of $a \Rightarrow b$, and so on.) We embody this meta-rule in the present system, for practical reasons.

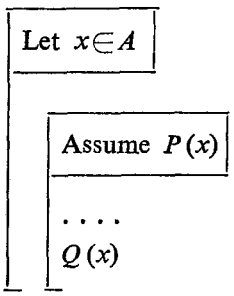
A. *Introduction rules*

Assumptions play an essential role in natural deduction. They are generally used with the purpose of simplifying in a natural manner the statement which has to be proved: a particular related statement is temporarily taken as an added datum, another statement, simpler than the original one, is the new object for proving. As soon as this aim is achieved, the assumption is "discharged", and the original statement has been proved.

This way of dealing with assumptions will be expressed in the notations used: sentences which are assumptions will be specially marked by a box; the range of

validity of an assumption will be marked by a vertical line starting from the left end of the box.

Thus doing it becomes apparent how the outer structure of a statement to be proved is reflected in its proof, as is often the case. For example, a proof of $\forall_{x\in A}[P(x) \Rightarrow Q(x)]$ will usually have the following shape:



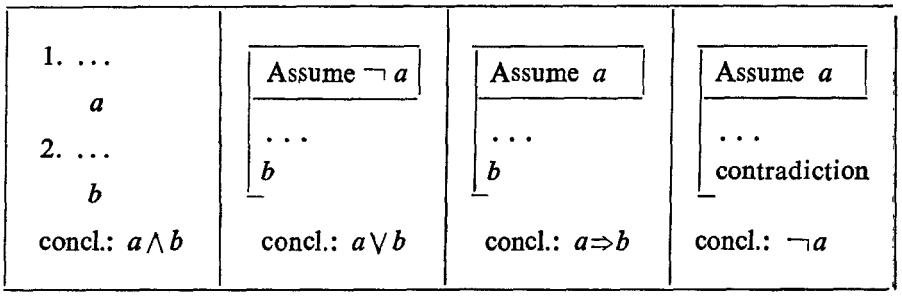
For presenting an outer proof structure in this manner it is desirable to organize a proof in such a way that validity ranges of assumptions are disjoint or nested: one should arrange these validity ranges in a block configuration as is known from programming languages.

From this example it may be seen that the sentence "Let $x\in A$" will appear as an assumption in our $\forall$-introduction rule. Our preference for assumptions rather than parameters in this rule is prompted by mathematical practice: in a proof of $\forall_{x\in A}[P(x)]$, the natural first step is: "Let $x\in A$".

It will be clear that the latter sentence is not an assumption in the proper sense, as it also introduces the variable x. There is, however, a strong analogy with "normal" assumptions of the kind "Assume p", notably with respect to validity and use. Therefore we shall all the same call "Let $x\in A$" an assumption, distinguishing this kind of assumption from the other by using the word "let" instead of "assume".

Our $\vee$-introduction rule deviates from the usual rules. Our argument for this is that the two "natural" proofs for $a \vee b$ look like a proof of an implication; for example: start with : "Assume $\neg a$" and derive a proof of b. Because our system is based on classical logic (see subsection C), the usual $\vee$-introduction rules are derived rules. (We confine ourselves to one rule for $\vee$-introduction and one for $\wedge$-introduction, the symmetry of $\vee$ and $\wedge$ being presupposed.)

Thus we propose the following standard proof schemes for introduction of $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$ and $\exists$, respectively:



Let $x \in A$ ... $P(x)$	... $t \in A$ $P(t)$
concl.: $\forall_{x \in A}[P(x)]$	concl.: $\exists_{x \in A}[P(x)]$

In applying any of these schemes one should insert (from above downwards) a sequence of sentences in the place of the dots, each sentence being justifiable in the sense explained before.

B. *Elimination rules*

We denote the legitimacy of a deduction of G from F by: $F \therefore G$. As elimination rules for $\wedge$, $\vee$, $\Rightarrow$, $\neg$, $\forall$ and $\exists$ we propose:

$a \wedge b \therefore a$

$a \vee b,\ a \Rightarrow c,\ b \Rightarrow c \therefore c$

$a \Rightarrow b,\ a \therefore b$

$\neg a,\ a \therefore$ contradiction

$\forall_{x \in A}[P(x)],\ t \in A \therefore P(t)$

$\exists_{x \in A}[P(x)],\ \forall_{x \in A}[P(x) \Rightarrow Q] \therefore Q$

In the $\exists$-elimination rule Q must not depend on x.

In a way, each elimination rule is the inverse of the corresponding introduction rule (cf. Prawitz [7, p.33]). There is, however, an essential difference in use between the two kinds of rules which, in our opinion, disturbs the symmetry: in principle, introduction rules give the general structure of a proof (cf. what has been mentioned in subsection A), elimination rules, however, are used for proceeding stepwise in the body of the proof. The difference in the notation of introduction and elimination rules, as shown above, reflects this asymmetry.

C. *Double negation rule*

Because logic, as it is generally used, is classical, we add the double negation rule:

$\neg\neg a \therefore a$

The absurdity rule (contradiction $\therefore a$) is now a derived rule.

The main difference between the logical system described above and the usual Gentzen-type systems are found in the $\forall$- and $\vee$-introduction rules. The block structure for validity ranges of assumptions is also present in the system of Kalish and Montague ([5]). The latter system employs an existential instantiation rule instead of the usual $\exists$-elimination rule (cf. the comment by Prawitz on this subject in Appendix C of [7]).

The system outlined above is suitable for tuition purposes. It has the advantages of natural deduction, both in setting up proofs and in understanding them. It also agrees closely with usual patterns of reasoning: a proof written with the aid of this system scarcely deviates from usual proofs, the differences being hardly more than boxes and validity lines.

On the other hand, as stated in the beginning of this section, formalization in the above system is not pushed very far. There are no formal devices for frequently occurring mathematical routines, like applying a theorem or a definition, justifying a deduction step, and so on. In the next section we shall describe how these sides of mathematical reasoning can be effectively formalized.

II. A system for natural reasoning

We shall describe a system with a wide range of applications and a high level of formalization. The system now to be proposed is natural in the sense that it is closely related to the usual way of reasoning and proving in mathematics. In the first instance, the system refers mainly to the nonlogical part of mathematics. However, rules of logic can be expressed and applied in the system. One may choose natural deduction as a basic for logic, in the manner of the previous section (as we do in II F), thus preserving the "natural" character of the system.

The system is directly derived from the "mathematical language Automath", designed by N.G. de Bruijn for rendering mathematical texts in a formal way (see [1]). Various versions of this language have been developed by de Bruijn, in cooperation with, among others, D.T. van Daalen, L.S. Jutting and J. Zucker (see [2]). Most of the features of these various versions will be present in the "system for natural reasoning", which we shall describe in this section.

A text formalized in such a system consists of a sequence of sentences, constructed one by one in accordance with the rules of the system, the "syntax". We shall not discuss the syntax rules in detail. For this we refer to the precise definitions of a few Automath systems in [2] or [3].

A mathematical text selected for being formalized in a system like the one at issue must not show any omission in its chain of reasoning; if necessary, it must be made complete. An appropriate "translation" of that text, i.e. a formalization in the system, will be complete as well, in the sense that every sentence can be mechanically verified as to correctness according to the syntax. The latter property obviously implies that one may attach a high degree of mathematical cogency to a text, which has been translated and verified in such a system.

A number of mathematical texts have actually been formalized in systems of this kind. For example: Jutting has translated a well-known mathematical text (see [4]), and the formalized text has been verified by means of a computer programme; Zucker formalized a part of classical real analysis (cf. [11]).

A. *Typing*

In mathematics one usually attaches types to objects; one says: x is a natural number, p is a proposition. In our system we incorporate a relation "s has type t" in a formal way, denoted as $s : t$.

We fix two basic terms: π and τ, representing the type of all propositions and the type of all "classes", respectively. For example: $(p \Rightarrow q) : \pi$ and $\mathbf{N} : \tau$. (Here $\mathbf{N}$ denotes the class of all natural numbers). A class like $\mathbf{N}$ can be the type of some term of lower rank, as, for instance, in the sentence $x : \mathbf{N}$. But a proposition like $p \Rightarrow q$ can likewise be the type of some term, viz. its proof, as we shall now explain.

It is common in mathematics to deal with proofs of propositions only at a meta-level. Contrary to this, we shall incorporate proofs as terms in our system, denoted and manipulated just as the other terms in the system. This idea is well-known (for references: see [11, p. 135]). It is based on the observation that a proof of a proposition results from a kind of "construction".

As type of a proof we take the proposition it proves; if t is a proof for $p \Rightarrow q$, we write $t : (p \Rightarrow q)$. Conversely, if $r : \pi$ and $t : r$ then (proof) t asserts (proposition) r.

By the above agreements concerning typing we obtain a hierarchical relation between terms of the system. Terms π and τ are (the only) representatives of the highest level of abstraction, to be assigned *degree* 0. Terms like $p \Rightarrow q$ and $\mathbf{N}$ belong to a lower level (degree 1); terms like x and t belong to the lowest level (degree 2). In the present system we restrict ourselves to these three levels.

There is a notable contrast between our relation : ("has type") and the set-theoretical relation $\in$ ("is element of"). In set-theory, an element may belong to different classes: $x \in \mathbf{N}$ implies $x \in \mathbf{R}$, since $\mathbf{N} \subset \mathbf{R}$. As to relation :, however, we impose *uniqueness of type*: each term of degree 1 or 2 has a fixed type. (For a remark on this uniqueness: see the following subsection.) Typing thereby becomes an unambiguous, effective procedure; this facilitates mechanical checking.

Thus, in the case that $\mathbf{N}$ and $\mathbf{R}$ have been introduced independently, "natural number x" cannot be considered as a real number by a direct embedding of $\mathbf{N}$ into $\mathbf{R}$. This has obvious disadvantages, like the necessity of some non-trivial embedding device; on the other hand, obscuring identifications are absent.

B. *Conversions*

We note the complicating circumstance that a term in the system may have different manifestations, being interchangeable by means of *conversions*. There are three kinds of basic conversions. The first results from the application of a definition to (part of) a term; this is called *definitional conversion* (for an example: see subsection D). The second concerns the application of a function to an argument; it is called *functional conversion* or β-conversion (see subsection E). The third is caused by the renaming of a certain variable in a few occurrences in a term, without disturbing the pattern of binding in the term; it will be called *renaming conversion* or α-conversion.

Conversions change a term in appearance, without changing its nature. Different appearances of one term, related by conversions, will be called *equivalent*.

The above implies that the "uniqueness of type", discussed in subsection A, should be understood modulo conversion.

C. *Assumptions*

In the natural deduction system of § I, assumptions appeared in two shapes: either in the simple version "Assume p", or in the more complex version "Let $x \in A$" (cf. I A). Since we regard proofs as terms, we may replace the former version by "Let t be a proof of proposition p". The latter version becomes "Let x be a term which has type A", in correspondence with our view upon typing. Formally, both versions of assumptions can be denoted, quite similarly, by the sentences $\boxed{t : p}$ and $\boxed{x : A}$, respectively, t and x being variables, p and A being terms. (An arbitrary assumption $\boxed{u : v}$ can be correctly interpreted by regarding the type of v.)

D. *Axioms, definitions, theorems*

We shall now describe how axioms, definitions and theorems can be incorporated.

Axioms (including basic notions) will be denoted by means of a double box. For example, in regarding N as a basic notion we obtain the sentence: $\boxed{\boxed{\mathrm{N} : \tau}}$. Then Peano's first axiom will be rendered by: $\boxed{\boxed{one : \mathrm{N}}}$. Axioms may contain one or more assumptions, like in Peano's second axiom, postulating a successor to every natural number; we may express this axiom by means of two sentences: $\boxed{x : \mathrm{N}}$ $\boxed{\boxed{s(x) : \mathrm{N}}}$. Here the assumption variable x returns in the latter sentence.

In such cases, when a sentence depends on an assumption variable, one may *instantiate*, i.e. (simultaneously) substitute a term for each occurrence of this variable in the sentence. It is then a natural requirement that the substituted term has an "appropriate" type. For example, from the last axiom one may infer that $s(one)$ has type N. Analogous rules hold in the case in which a sentence depends on more than one variable.

Definitions will be written as in the following examples:

$two := s(one) : \mathrm{N}, \qquad three := s(two) : \mathrm{N},$

$\boxed{y : \mathrm{N}}\ plustwo\ (y) := s\,(s\,(y)) : \mathrm{N}.$

In the last example the definition consists of two sentences, the latter depending on the former.

The three above examples concern definitions of terms of degree 2. It is also possible to write a sentence containing a definition of a term of degree 1; such a term defines a "class", a proposition or a predicate.

For proofs of theorems we use the same notation as for definitions that concern terms of degree 2. We justify this policy with the following remark: a proof of a theorem *th* (where $th:\pi$) fixes a term p with $p : th$, while a definition of an "object" belonging to a "class" *cl* (where $cl:\pi$) fixes a term b with $b : cl$.

We shall show in an example how a theorem can be expressed (and proved). Suppose that the relation equality for natural numbers ($=_N$) is given as a basic notion by $\boxed{x:\mathbf{N}}$ $\boxed{y:\mathbf{N}}$ $\boxed{\boxed{(x=_N y):\pi}}$. Let reflexivity of $=_N$ be given by axiom:

$$\boxed{x:\mathbf{N}} \quad \boxed{\boxed{refis\,(x):(x=_N x)}}.$$

Now a proof of the theorem *plustwo* $(one)=_N$ *three* can be expressed by:

proof 1 $:=$ *refis* $(s\,(s\,(one))):(plustwo\,(one)=_N three)$.

At first sight this seems incorrect, because the axiom for reflexivity yields the relation

$$refis\,(s\,(s\,(one))):(s\,(s\,(one))=_N s\,(s\,(one))),$$

by substituting $s\,(s\,(one))$ for x. But by means of definitional conversion (see subsection B) and instantiation we may change $(plustwo\,(one)=_N three)$ into $(s\,(s\,(one))=_N (s\,(s\,(one)))$, by applying the definitions given above of *plustwo* (*one*), *three* and *two*.

As some of our examples showed, axioms and definitions may consist of more than one sentence, all but the last being assumptions. This may also be the case with (proved) theorems. Such an initial sequence of assumptions is called a *context* for the axiom, definition or theorem at issue; the assumption variables of a context may occur in the final sentence. The interdependence may even be stronger: each assumption variable in a context may occur in "type-parts" of assumptions which follow in that context. See, for instance, the axiom for the double negation rule, given in subsection F.

E. *Functions*

Functional abstraction and application form part of the system. For functional abstraction we use an adapted lambda-calculus notation, demonstrated by the following definition: $idfun := [\lambda x : \mathbf{N}]x : [\lambda x : \mathbf{N}]\,\mathbf{N}$. Here $[\lambda x : \mathbf{N}]x$ is the identity function for natural numbers; the type of this function, $\mathbf{N}^{\mathbf{N}}$, is denoted by the "type-valued function" $[\lambda x : \mathbf{N}]\mathbf{N}$. Application of funtcion f to argument x is denoted by $\{x\}f$. A motivation of the unusual order of function and argument is given in [6, p. 11—12].

A natural requirement regarding functions is that an argument of a function must have a type equivalent (in the sense explained in subsection B) to the domain of that function.

For functions and arguments the laws of *functional conversion* (also called β-conversion) hold, allowing for example the conversion of $\{A\}[\lambda x : B]C$ into ${}^{x}_{A}C$, i.e. the result of substituting term A for all free occurrences of x in term C. (We gave a general description of conversions in subsection B; q.v.)

Example: application of *idfun* to *two* yields $\{two\}\,idfun$, which is equivalent to $\{two\}[\lambda x : \mathbf{N}]x$ by definitional conversion. Then by functional conversion we may change the latter into *two*. Hence, $\{two\}\,idfun$ and *two* are equivalent: they are both "appearances of the same term".

F. *Deduction rules*

As stated before, logical rules are not primitive in the present system: one may choose one's own logical basis. We shall show how one may incorporate rules for natural deduction by means of axioms and definitions. In this respect the formal correspondences between $\Rightarrow$ and $\forall$ on one hand and functional abstraction on the other, can be successfully exploited.

For example, the "meaning" of $p \Rightarrow q$ is that for every proof of proposition p we may produce a proof of proposition q. This is a functional relation. Hence it seems natural to define $p \Rightarrow q$ as type-valued function $[\lambda x : p]q$. Then application of modus ponens can be simply effectuated by functional application (and a few conversions):

$$\boxed{s : p} \quad \boxed{t : (p \Rightarrow q)} \quad modponapp\,(s, t) : = \{s\}\,t : q\,.$$

The "meaning" of $\forall_{x \in A}[P(x)]$ is that to every x in A, a proof of $P(x)$ can be attached. This is again a functional relation. So one may define $\forall_{x \in A}[P(x)]$ as the type-valued function $[\lambda x : A]P(x)$. The role of the $\forall$-elimination rule will again be taken over by the rule of functional application.

Contradiction may be introduced as basic notion:

$\boxed{\boxed{contradiction : \pi}}$. Then $\neg p$ can be defined as $p \Rightarrow contradiction$. Now $\neg$-elimination becomes a special case of modus ponens.

The double negation rule has to be expressed by means of an axiom as follows:

$$\boxed{p : \pi} \quad \boxed{n : \neg(\neg p)} \quad \boxed{\boxed{doubneg\,(p, n) : p}}\,.$$

The logical constants $\wedge$, $\vee$, $\Leftrightarrow$ and $\exists$ can be defined in terms of $\Rightarrow$, $\neg$ and $\forall$, in the usual way. The introduction and elimination rules for $\wedge$, $\vee$ and $\exists$ can subsequently be derived as theorems.

III. *A uniform system*

In the system of § II there exists a strong correspondence between contexts ("sequences of assumptions", see II D) and functional abstractions. For example, the axiom $\boxed{x : \mathbf{N}} \quad \boxed{\boxed{s(x) : \mathbf{N}}}$, using a context consisting of a single assump-

tion, could be replaced by $\boxed{\boxed{S : [\lambda x : \mathrm{N}]\,\mathrm{N}}}$, using functional abstraction. The role of "instantiation", substitution of a term for an assumption variable (cf. II D), will be taken over by functional application: $s(two)$ becomes $\{two\}S$.

We shall now propose a uniform system, developed by de Bruijn and Nederpelt (see [6]), wherein, to begin with, all assumptions are written as abstractions of the form $[\lambda k : L]$. We denote axioms and basic notions as abstractions, too, because one may regard these as being assumptions with unlimited validity range. For example, the above axiom will be written as $[\lambda S : [\lambda x : \mathrm{N}]\mathrm{N}]$.

Definitions obtain a uniformized shape as well, because instead of, e.g., $z := A : B$, A and B being terms, we write $\{A\}[\lambda z : B]$. Here variable z is defined as being functional application term A, both having type B. The role of definitional conversion (replacement of z by A) is taken over by functional conversion: $\{A\}[\lambda z : B]C$ is equivalent to ${}^{z}_{A}C$.

It this system we write theorems together with their proofs, in a manner similar to that in which definitions are written. For example: $\{D\}[\lambda z : E]$ may express theorem E and its proof D, z being a name for the proof.

In the case in which a definition or a proved theorem depends on a non-empty context, the formulation in the present system is somewhat more complicated than suggested above.

By means of uniformation, such as above, we obtain a simplified system, which is a typed lambda-calculus with lambda-structured types and two constants: π and τ. This typed lambda-calculus, which we call Λ, can be regarded as a model for "systems of natural reasoning" like that described in § II, in the sense that it gives a simple and uniform framework for such systems.

As an example we give the reformulation of the theorem $plustwo(one) =_N three$ discussed in § II. In Λ this theorem becomes a single line, containing all needed information:

$$[\lambda \mathrm{N} : \tau][\lambda S : [\lambda x : \mathrm{N}]\mathrm{N}][\lambda ONE : \mathrm{N}]\{\{ONE\}S\}[\lambda TWO : \mathrm{N}]$$
$$\{\{TWO\}S\}[\lambda THREE : \mathrm{N}]\{[\lambda y : \mathrm{N}]\{\{y\}S\}S\}[PLUSTWO,$$
$$[\lambda y : \mathrm{N}]\mathrm{N}][\lambda ISN : [\lambda x : N][\lambda y : \mathrm{N}]\pi][\lambda REFIS : [\lambda x : \mathrm{N}]$$
$$\{x\}\{x\}ISN]\{\{ONE\}PLUSTWO\}\{THREE\}ISN.$$

The proof of this theorem looks similar, but for the last part

$\{\{ONE\}PLUSTWO\}\{THREE\}ISN$, which reads:

$\{\{\{ONE\}S\}S\}REFIS.$

We do not uniformize π and τ into one constant, as is done in [6], since we wish to prevent assertions concerning propositions from having consequences for "classes", and vice versa. The double negation rule, for example, would in that case imply some form of the axiom of choice (cf. [11, p. 141]).

We shall now give a precise definition of Λ as being a class of terms in a typed lambda-calculus.

The *alphabet* under consideration consists of *constants* π and τ, an infinite number of *variables*: $x, y, \ldots$ and the *improper symbols* [,], {, }, λ and : .

Terms are recursively defined by:

(1) π and τ are terms; each variable x is a term.

(2) If A and B are terms and if x is a variable, then $[\lambda x : A]B$ and $\{A\}B$ are terms.

The relation: K is a *subterm* of L is the reflexive and transitive relation generated by:
A and B are subterms of $[\lambda x : A]B$ and of $\{A\}B$.

Type$_K$ is a partial function from the set of subterms occurring in term K to the set of all terms, which function is recursively defined by:

(1) If variable x occurs in K as a subterm and x is bound by $[\lambda x : A]$ in K, then $\mathrm{Type}_K(x) = A$.

(2) (monotony:) If $[\lambda y : A]B$ is a subterm of K, if $\mathrm{Type}_K(B)$ is defined and if $\mathrm{Type}_K(B) = C$, then $\mathrm{Type}_K([\lambda y : A]B) = [\lambda y : A]C$. Under analogous conditions: $\mathrm{Type}_K(\{A\}B) = \{A\}C$.

Degree$_K$ is a partial function from the set of subterms occurring in term K to the set of the non-negative integers, which function is recursively defined by:

(1) If subterm A of K ends in τ or π, then $\mathrm{degree}_K(A) = 0$.

(2) If subterm A of K ends in variable x, bound by $[\lambda x : B]$, and if $\mathrm{degree}_K(B)$ is defined, then $\mathrm{degree}_K(A) = \mathrm{degree}_K(B) + 1$.

(In Λ there is no upper bound for the values of the degree function.)

Bound terms are terms without free variables.

(In bound terms all subterms have a degree and all subterms not ending in τ or π have a type.)

α-reduction, denoted $>_\alpha$, is the monotonous relation generated by $[\lambda x : B]\, C >_\alpha [\lambda y : B]^x_y C$, with the usual restriction that the pattern of binding may not be disturbed.

β-reduction, denoted $>_\beta$, is the monotonous relation generated by $\{A\}[\lambda x : B]C >_\beta {}^x_A C$. (In substituting A for x, variables must be renamed in the usual way, in order to prevent "clash of variables".)

Reduction, denoted $>$, is the reflexive and transitive closure of both α- and β-reduction. If $K > L$, then L is called a *reduct* of K. (One may consider reduction as "one-way conversion"; cf. § II B and E.)

Legitimate terms are bound terms K with the following property: For each subterm of K of the form $\{A\}B$ there exist C and D with the properties that $\mathrm{Type}^\gamma_K(B) > [y, C]D$ and that $\mathrm{Type}_K(A)$ and C have a common reduct; here $\gamma = \mathrm{degree}\ (B)$ and Type^γ_K is Type_K iterated γ times, which iteration is defined in the natural way.

Now Λ is defined as the set of all legitimate terms.

The limitation to legitimate instead of bound terms has two reasons. The first is of a intuitive nature: it is a natural requirement for a system, close to mathematical practice, that arguments A may only be related to terms B with an appropriate functional character. That is to say, B must, in a sense, be a func-

tion with a certain domain C. Moreover, argument A must be an object belonging to this domain C.

A second reason is that function applications (β-reductions) to bound terms may bring about a non-terminating process, just as in ordinary λ-calculus. In restricting oneself to legitimate terms this is impossible (see following theorem (3)).

We conclude with four theorems valuable for theoretical purposes:

(1) *Church-Rosser property* or diamond property: If $A > B$ and $A > C$, then B and C have a common reduct.

(2) *Normalization*: Every term in Λ has a normal form (i.e. a reduct to which no β-reduction can be applied), which is effectively computable; this normal form is unique but for α-reduction.

(3) *Strong normalization*: For no A in Λ is there an infinite reduction sequence $A >_\beta A_1 >_\beta A_2 >_\beta \ldots$.

(4) *Closure*: If A is in Λ and $A > B$, then B is in Λ.

For proofs of these theorems: see [6] and [3].

References

[1] N.G. de BRUIJN, *The mathematical language Automath*, its usage, and some of its extensions. Symposium on Automatic Demonstration, Versailles, Dec. 1968. Lecture Notes in Mathematics, vol. 125, pp.29—61, Berlin, 1970.

[2] D.T. van DAALEN, *A description of Automath and some aspects of its language theory*. Proceedings of the Symposium on APL, ed. P. Braffort, Paris, 1974.

[3] D.T. van DAALEN, *The language theory of Automath*. Thesis, Eindhoven, in preparation.

[4] L.S. van BENTHEM JUTTING, *Checking Landau's "Grundlagen" in the Automath system*. Thesis, Eindhoven, 1977.

[5] D. KALISH and R. MONTAGUE, *Logic, techniques of formal reasoning*. New York, 1964.

[6] R.P. NEDERPELT, *Strong normalization in a typed lambda-calculus with lambda-structured types*. Thesis, Eindhoven, 1973.

[7] D. PRAWITZ, *Natural deduction, a proof-theoretical study*. Stockholm Studies in Philosophy 3, Stockholm, 1965.

[8] D. PRAWITZ, *Ideas and results in proof theory*. Proceedings of the Second Scandinavian Logic Symposium, ed. J.E. Fenstad, Amsterdam, 1971.

[9] W.V. QUINE, *Methods of logic*. London, 1952.

[10] P. SUPPES, *Introduction to logic*. Princeton, 1957.

[11] J. ZUCKER, *Formalization of classical mathematics in Automath*. Colloque International de Logique, Clermont-Ferrand, July 1975. Centre National de la Recherche Scientifique.