MATHEMATICAL CENTRE TRACTS 83

CHECKING LANDAU'S "GRUNDLAGEN" IN THE AUTOMATH SYSTEM

L.S. VAN BENTHEM JUTTING

MATHEMATISCH CENTRUM AMSTERDAM 1979

MS Classification (1980): 03-04,03B30,03B40,68-04,69F05,68F30

ISBN 90 6196 147 5

Contents page

PREFACE

by

N.G. de Bruijn

The Automath project was conceived early 1967 in Eindhoven, the Netherlands. Seemingly, the project was quite modest in its claims, since it intended to achieve a thing that many people thought to present no essential difficulties: presenting mathematics in such a precise formalized fashion that, the principles of the language being given, even a machine could establish correctness. The style in which this should be done was clear from the start: it should follow ordinary mathematical reasoning, both in words and in formulas, line by line, without changing the rôles which arguments, definitions, abbreviations, theorems, lemmas play in the original text, and without replacing the usual human verification by an essentially different process.

On the other hand the project was quite pretentious, claiming that is could practice what it preached. It claimed that formalization and verification could really be carried out for substantial portions of mathematics, with a language more closely resembling intuitive mathematics then other present day formalizations, and fit for describing a large variety of concepts and situations.

In order to substantiate this claim, and to acquire experience, it was decided in 1968 to undertake the translation of a complete book, for which Landau's "Grundlagen der Analysis" were chosen. L.S. Jutting, the first collaborator on the project, would be the one to carry out this task.

The work, including machine verification, was finished in September 1975. The complete text is presented in a 500 page report (L.S. Jutting, A translation of Landau's "Grundlagen" in AUTOMATH). Some fragments are reproduced here (appendices 3, 4 and 7).

The present book intends to render an account of the translation. It reports on the difficulties that were encountered and the way these were taken care of. It also describes the nature of AUTOMATH and its usage. Moreover, by permission of D.T. van Daalen, his paper on the formal definition of the language is reprinted here (appendix 1). What this book does not show much of is Mr. Juttings persistence in carrying this job to its end. Whether this work is important or interesting are matters of opinion and taste, but it is hard to deny that it has a certain historic value: never before has a substantial piece of mathematics been presented on a comparable level of completeness and precision.

0. INTRODUCTION

In this chapter a brief description of the AUTOMATH project is given, and the place of the present work within this project is indicated.

0.0. The AUTOMATH languages

The languages of the AUTOMATH family are formal languages, in which large parts of mathematics can be efficiently formalized. Texts in these languages can be checked mechanically (i.e. by a computer). A text is verified line by line, and the checking does not only cover syntactical correctness of the expressions occurring in a line, but also its mathematical validity, which includes the correctness of references to previous lines. Correct AUTOMATH texts may therefore be interpreted*) to represent correct mathematics.

The structure of these languages, based on natural deduction, is closely related to the structure of common intuitive reasoning. Hence mathematical discourses in an informal language can be translated into an AUTOMATH language without too much trouble.

At the moment a number of mutually related languages exist satisfying the above specifications. For several of these languages, verifying computer programs are currently operational; for others, such programs are still in an experimental stage.

0.1. The AUTOMATH project and its motivation

The object of the AUTOMATH project has been to develop languages as described above, and to make verifying computer programs for these languages. It was initiated some ten years ago by N.G. de Bruijn, who also conceived the fundamentals of the AUTOMATH languages. Since then a number of mathematicians have been working on the project, providing AUTOMATH with a language theory, writing verifying programs for AUTOMATH languages, producing texts in these languages, and applying the verifying programs to these texts.

There were several reasons for initiating such a project, of which we mention the following:

---

*) In discussing an AUTOMATH text I will call the intended meaning (in formal or informal mathematics) of this text its *interpretation*, and I will say that this meaning is *represented* in the AUTOMATH text.

i) Mechanical verification will increase the reliability of certain kinds of proofs. A need for this may be felt where a proof is extremely long, complicated and tedious, and where it is difficult to break it down into intuitively plausible partial results; or where in proofs results of others are used, so that misinterpretations become possible.

ii) Mechanically verifiable languages set a standard by which informal language may be measured, and may thereby have an influence on the use of language in mathematics generally.

iii) The use of such languages gives an insight into the structure of mathematical texts, and makes it possible to compare the complexity, in several respects, of mathematical concepts and proofs. As a consequence projects of this kind may have in the long run a favourable influence on the teaching of mathematics.

A further motive, for the author, was that the work involved in the project appealed to him.

More information on the AUTOMATH project, its objectives, motivation and history can be found in [dB].

0.2. The book translated

At an early stage of the AUTOMATH project the need was felt to translate an existing mathematical text into an AUTOMATH language, first, in order to acquire experience in the use of such a language, and secondly, to investigate to what extent mathematics could be represented in AUTOMATH in a natural way.

As a text to be translated, the book "Grundlagen der Analysis" by E. Landau [L] was chosen. This book seemed a good choice for a number of reasons: it does not presuppose any mathematical theory, and it is written clearly, with much detail and with a rather constant degree of precision.

For a short description of the contents of Landau's book see 2.0.

0.3. The language of the translation

The language into which Landau's book has been translated is AUT-QE. A detailed description and a formal definition of this language is given in [vD]. As this paper is fundamental to the following monograph and not easily obtainable, it has been added as appendix 1. I will use the notations introduced there whenever necessary. Where in the following text a concept

introduced in [vD] is used for the first time, it will be displayed in italics, with a reference to the section in [vD] where it occurs.

The language of the translation differs from the definition in [vD] in one respect, viz. the division of the text into *paragraphs* [vD, 2.16]. By this device the strict rule that all *constants* [vD, 2.6, 5.4.1] in an AUT-QE *book* [vD, 2.13.1, 5.4.4] should be different is weakened to the more liberal rule that all constants in one paragraph have to differ. Now, in a *line* [vD, 2.13, 5.4.4], reference to constants defined in the paragraph containing that line is as usual, while reference to constants defined in other paragraphs is possible by a suitable reference system. For a more detailed description of the system of paragraphing, see appendix 2.

In contravention of the rules for the shape and use of names in AUT-QE, we will in examples in the following text not restrict ourselves to alphanumeric symbols, and occasionally we use infix symbols. (Of course, in the actual translation of Landau's book, these deviations from proper AUT-QE do not occur.)

1. PREPARATION

In this chapter the logic which Landau presupposes is analysed and its representation in AUT-QE is described.

1.0. The presupposed logic

In his "Vorwort für den Lernenden" Landau states: "Ich setze logisches Denken und die deutsche Sprache als bekannt voraus". Clearly, in the translation AUT-QE should be substituted for "die deutsche Sprache". The proper interpretation of "logisches denken" must be inferred from Landau's use of logic in his text.

This appears to be a kind of informal second (or higher) order predicate logic with equality. In the following some characteristics of Landau's logic will be discussed, and illustrated by quotations from his text.

i) Variables have well defined ranges which are not too different from *types* [vD, 2.2] in AUT-QE. Cf.:
   - On the first page of "Kapitel 1": "Kleine lateinische Buchstaben bedeuten in diesem Buch, wenn nichts anderes gesagt wird, durchweg natürliche Zahlen".
   - In "Kapitel 2, § 5": Grosze lateinische Buchstaben bedeuten durchweg, wenn nichts anderes gesagt wird, rationale Zahlen".

ii) Predicates have restricted domains, which again can be interpreted as types in AUT-QE. Cf.:
   - "Satz 9: Sind x und y gegeben, so liegt genau eine der Fälle vor:
     1) x = y.
     2) Es gibt ein u mit x = y + u ..." etc.
     It is clear that u (being a lower case letter) is a natural number, or u $\underline{E}$ nat.
   - "Definition 28: Eine Menge von rationalen Zahlen heiszt ein Schnitt, wenn...".
     Here it is apparent that being a "Schnitt" is a predicate on the type of sets of rational numbers.

iii) When, for a predicate P, it has been shown that a unique x exists for which P holds, then "the x such that P" is an object. Cf.:
   - "Satz 4, zugleich Definition 1: Auf genau eine Art läszt sich jedem Zahlenpaar x,y eine natürliche Zahl, x + y genannt, so zuordnen. dasz... . x + y heiszt die Summe von x und y".

- "Satz 101: Ist $X > Y$ so hat $Y + U = X$ genau eine Lösung U.
  Definition 23: Dies U heiszt $X - Y$".

iv) The theory of equivalence classes modulo a given equivalence relation, whereby such classes are considered as new objects, is presupposed by Landau. Cf.:

- The text preceding "Satz 40": "Auf Grund der Sätze 37 bis 39 zerfallen alle Brüche in Klassen, so dasz $\frac{x_1}{x_2} \sim \frac{y_1}{y_2}$ dann und nur dann, wenn $\frac{x_1}{x_2}$ und $\frac{y_1}{y_2}$ derselben Klasse angehören".
- "Definition 16: Unter eine rationale Zahl versteht mann die Menge aller einem festen Bruch äquivalenten Brüche (also eine Klasse im Sinne des § 1)".

v) The concepts "function" and "bijective function" are vaguely described. Cf.:

- "Satz 4" (see iii) above).
- "Satz 274: Ist $x < y$ so können die $m \leq x$ nicht auf die $n \leq y$ eindeutig bezogen werden".
- "Satz 275: Es sei x fest, f(n) für $n \leq x$ definiert. Dann gibt es genau ein für $n \leq x$ definiertes $g_x(n)$ mit folgenden Eigenschaften..." followed by the "explanation": "Unter definiert verstehe ich: als komplexe Zahl definiert". This explanation might be interpreted to indicate the typing of the functions f and g.

vi) Landau defines and uses partial functions. Cf.:

- "Definition 14: Das beim Beweise des Satzes 67 konstruierte spezielle $\frac{u_1}{u_2}$ heiszt $\frac{x_1}{x_2} - \frac{y_1}{y_2}$ ...". Here the construction, and therefore the definition, only applies if $\frac{x_1}{x_2} > \frac{y_1}{y_2}$.
- "Definition 56: Das Y des Satzes 204 heiszt $\frac{\Xi}{H}$ ". This definition depends upon the assumption $H \neq 0$.
- "Definition 71", where Landau states explicitly: "Nicht definiert ist x^n also lediglich für $x = 0$, $n \leq 0$".
- "Satz 155: Beweis: II) Aus $X > Y$ folgt $X = (X - Y) + Y$".
- "Satz 240: Ist $y \neq 0$ so ist $\frac{x}{y} . y = x$".
- "Satz 291: Es sei $n > 0$ oder $x_1 \neq 0$, $x_2 \neq 0$. Dann ist $(x_1 . x_2)^n = x_1^n . x_2^n$ ".

In these last three examples we see *"generalized implications"*: the terms occurring in the consequent are meaningful only if the antecedent is taken to be true. A similar situation will be encountered in vii).

vii) Definitions by cases, sometimes of a complicated nature, are used. Cf.:

- "Definition 52:

$$\Xi + H = \begin{cases} -(|\Xi| + |H|) & \text{wenn } \Xi < 0,\ H < 0. \\ \left.\begin{cases} |\Xi| - |H| \\ 0 \\ -(|H| - |\Xi|) \end{cases}\right\} & \text{wenn } \Xi > 0,\ H < 0, \begin{cases} |\Xi| > |H|. \\ |\Xi| = |H|. \\ |\Xi| < |H|. \end{cases} \\ H + \Xi & \text{wenn } \Xi < 0,\ H > 0. \\ H & \text{wenn } \Xi = 0. \\ \Xi & \text{wenn } H = 0". \end{cases}$$

- "Definition 71:

$$x^n = \begin{cases} \prod_{k=1}^{n} x & \text{wenn } n > 0. \\ 1 & \text{wenn } x \neq 0,\ n = 0. \\ \frac{1}{x^{|n|}} & \text{wenn } x \neq 0,\ n < 0. \end{cases}$$

Notice that in these two definitions, in some of the cases the definiens is not defined when the corresponding condition does not hold, (*"generalized definition by cases"*), and also that, in some cases, there is in the definiens a reference to the definiendum.

viii) In his text Landau only occasionally mentions predicates and relations; usually he refers to sets. Cf.:

- "Axiom 5: Es sei M eine Menge natürlicher Zahlen mit den Eigenschaften:

  I) 1 gehört zu M.

  II) Wenn x zu M gehört, so gehört x' zu M.

  Dann umfaszt M alle natürlichen Zahlen".

- "Satz 2: x' ≠ x. Beweis: M sei die Menge der x, für die dies gilt..".

However, in the text preceding "Definition 26":

- "Da =, >, <, Summe und Produkt den alten Begriffen entsprechen...".

ix) Landau considers (ordered) pairs of objects. In chapter 2 the components of such pairs remain clearly visible in their names: he does not refer to "the pair x with components x_1 and x_2", but only to "the pair x_1,x_2". Nevertheless it is clear from his words that he considers such a pair as one object. Cf.:

- "Definition 7: Unter einem Bruch $\frac{x_1}{x_2}$ versteht man das Paar der natürlichen Zahlen x_1,x_2 (in dieser Reihenfolge)".
- "Definition 8: $\frac{x_1}{x_2} \sim \frac{y_1}{y_2}$ wenn $x_1y_2 = y_1x_2$".

In chapter 5 however, variables for pairs *are* used. Cf.:

- "Definition 57: Eine komplexe Zahl ist ein Paar reëller Zahlen Ξ_1,Ξ_2 (in bestimmter Reihenfolge). Wir bezeichnen die komplexe Zahl mit $[\Xi_1,\Xi_2]$".

This definition is immediately followed by

- "Kleine deutsche Buchstaben bedeuten durchweg komplexe Zahlen".

The two notations are linked in the following way:

- "Definition 60: Ist $x = [\Xi_1,\Xi_2]$, $y = [H_1,H_2]$, so ist $x + y = [\Xi_1 + \Xi_2, H_1 + H_2]$".

x) Finally it should be pointed out that some of Landau's proofs and remarks tend to a kind of intuitive reasoning which is not easily represented in a formal system.

A first example of this is the treatment of equality in "Kapitel 1, § 1".

- "Ist x gegeben und y gegeben, so sind entweder x und y dieselbe Zahl; das kann man auch $x = y$ schreiben; oder x und y nicht dieselbe Zahl; das kann men auch $x \neq y$ schreiben.

  Hiernach gilt aus rein logischen Gründen:

  1) $x = x$ für jedes x.

  2) Aus $x = y$ folgt $y = x$.

  3) Aus $x = y$, $y = z$ folgt $x = z$".

Here it seems that Landau derives the properties of equality from reflection on the properties of a mathematical structure. They are not theorems or axioms but intuitively true statements. Substitutivity of equal objects, though used frequently in the proofs of subsequent theorems, is never mentioned.

Other examples of proofs with intuitive components may be found where Landau, in a glance, takes in a complex logical situation. Cf.:

- "Satz 16: Aus $x \leq y$, $y < z$ oder $x < y$, $y \leq z$ folgt $x < z$. Beweis: Mit dem Gleichheitszeichen in der Voraussetzung klar; sonst durch Satz 15 erledigt".
- "Satz 20: Aus $x + z > y + z$ bzw. $x + z = y + z$ bzw. $x + z < y + z$ folgt $x > y$ bzw. $x = y$ bzw. $x < y$.
  Beweis: Folgt aus Satz 19 da die drei Fälle beide Male sich ausschlieszen und alle Möglichkeiten erschöpfen".

A somewhat different example, which involves what might be called "metalogic", is the text preceding "Definition 26", where it is indicated how a number of theorems might be proved, without actually proving them. I will return to this in 2.1 viii).

1.1. The representation of logic in AUT-QE

The logic considered by Landau to be "logisches Denken", as described in the previous section, has been formalized in the first part of the AUT-QE book, called "preliminaries", which, unlike the other parts, does not correspond to an actual chapter of Landau's book.

A possible way of coding logic in AUT-QE has been described in [vD, 3,4]. In addition to this description we stress a few points on the interpretation of AUT-QE lines [vD, 2.13, 5.4.4]. Adopting the terminology introduced in [Z] we shall call expressions of the form $[x_1,\alpha_1]...[x_k,\alpha_k]$ type (with $k \geq 0$) (i.e. t-expressions of degree 1) 1t-*expressions* and expressions of the form $[x_1,\alpha_1]...[x_k,\alpha_k]$ prop (again with $k \geq 0$) 1p-*expressions*. Expressions having 1t- and 1p-expressions as their types, will be called 2t-*expressions* and 2p-*expressions*, respectively. Finally, 3t- and 3p-*expressions* have 2t- and 2p-expressions as their types.

Now a 2t-expression will be used to denote a type (or "class"). If its type is an *abstraction expression* [vD, 2.8, 5.4.2] then it denotes a type of functions. A 2p-expression denotes a proposition or a predicate. A 3t-expression denotes an object (of a certain type) and a 3p-expression a proof (of a certain proposition).

The interpretation of an AUT-QE line having a certain shape (EB-*line*, PN-*line* or *abbreviation line* [vD, 2.13, 5.4.4]) will depend on its *category part* [vD, 2.13.1] being a 1t-, 1p-, 2t- or 2p-expression. So we arrive at the following refinement of the scheme in [vD, 4.5].

Shape of the line: Category-part

	1t-expression	1p-expression	2t-expression	2p-expression
EB-line	introduces a type variable	introduces a proposition or predicate variable	introduces an object variable (of the stated type)	introduces the stated proposition as an assumption
PN-line	introduces a primitive type constant	introduces a primitive proposition or predicate constant	introduces a primitive object (of the stated type)	introduces the stated proposition as an axiom
Abbreviation line	defines a type in terms of known concepts	defines a proposition or predicate in terms of known concepts	defines an object (of the stated type) in terms of known concepts	proves the stated proposition as a theorem

In the above scheme it is apparent that, if the category part of a line is a 2p-expression, then the interpretation of that line is an assertion. But also if the category part is a 2t-expression α the interpretation has an assertional aspect; the line does not only introduce a new name for an object (as a variable, or a primitive or defined constant) but also asserts that this object has the type α.

1.2. Account of the PN-lines

Here I will give a survey of the primitive concepts and axioms (PN-lines) occurring in the preliminary AUT-QE text. A mechanically produced list of these axioms appears as appendix 3. In this list the PN-lines appear numbered. References in parentheses below will refer to these numbers.

i) Axioms for contradiction.
Contradiction is postulated as a primitive proposition (1), the double negation law as an axiom (2).

ii) Axioms for equality.

Given a type S, equality is introduced as a primitive relation on S (3), with axioms for reflexivity (4) and for substitutivity (5) (i.e. if $x=y$, and if P is a predicate on S which holds at x, then P holds at y). Moreover there is an axiom stating extensionality for functions (8).

The notion of equality so introduced is called *book-equality* (cf. [vD, 3.6]) in contrast to *definitional equality* of expressions ([vD, 2.12, 5.5.6]).

iii) Axioms for individuals.

Given a type S, a predicate P on S, and a proof that P holds at a unique $x \underline{E} S$, the object ind (for individual) is a primitive object (6), to be interpreted as "the x for which P holds". An axiom states that ind satisfies P (7).

iv) Axioms for subtypes.

Given a type S and a predicate P on S, the type OT (for own-type, i.e. the subtype of S associated with P) is a primitive type (9). For $u \underline{E} OT$ we have a primitive object $\mathsf{in}(u) \underline{E} S$ (10), and an axiom stating that the function $[u,OT]\mathsf{in}(u)$ is injective (12). Moreover there are axioms to the effect that the images under this function are just those $x \underline{E} S$ for which P holds ((11) and (12)).

v) Axioms for products (of types).

Given types S and T the type $\mathsf{pairtype}$ (the type of pairs (x,y) with $x \underline{E} S$ and $y \underline{E} T$) is introduced as a primitive type (14). For $p \underline{E} \mathsf{pairtype}$ we have the projections $\mathsf{first}(p) \underline{E} S$ and $\mathsf{second}(p) \underline{E} T$ as primitive objects ((16) and (17)), and conversely, for $x \underline{E} S$ and $y \underline{E} T$ we have $\mathsf{pair}(x,y)$ as a primitive object in $\mathsf{pairtype}$ (15). Next there are three axioms stating that $\mathsf{pair}(\mathsf{first}(p),\mathsf{second}(p))=p$, $\mathsf{first}(\mathsf{pair}(x,y))=x$ and $\mathsf{second}(\mathsf{pair}(x,y))=y$ (where = refers to book-equality as introduced in ii)) ((19), (20) and (21)).

(Note: If a type U containing just two objects is available, and if S is a type, the type of pairs (x,y) with $x \underline{E} S$ and $y \underline{E} S$ may be defined alternatively as the function type $[x,U]S$. In the translation this was done at the end of chapter 1, where we took for U the subtype of the naturals ≤ 2. Therefore the pairing axioms as described above were not used in the actual translation.)

vi) Axioms for sets.

Given a type S, the type set (the type of sets of objects in S) is introduced as a primtive type (21), and the element relation as a primitive relation (22). Given a predicate P on S, there is a primitive object $\mathsf{setof}(P)$ $\underline{E}$ set (denoting the set of $x \underline{E} S$ satisfying P) (23), and there are two axioms to the effect that P holds at x iff x is an element of $\mathsf{setof}(P)$ ((24) and (25)).

These can be viewed as comprehension axioms for S. (As sets contain only objects of one type, such axioms will not give rise to Russell-type paradoxes.)

Finally extensionality for sets is stated as an axiom (26).

The axioms for sets permit "higher-order" reasoning in AUT-QE, since quantification over the type set is possible.

1.3. Development of concepts and theorems in Landau's logic

Here we give a sketch of the development of the logic in [L] from the axioms described in the previous section.

Starting from the axioms for contradiction, the development of classical first order predicate calculus is straightforward. In this development more then usual attention has been paid to mutual exclusion: $\neg(A \wedge B)$, and trichotomy: $(A \vee B \vee C) \wedge (\neg(A \wedge B) \wedge \neg(B \wedge C) \wedge \neg(C \wedge A))$, because these concepts are used frequently by Landau in discussing linear order.

The properties of equality, e.g. symmetry, transitivity, and substitutivity for functions (i.e. if $x=y$ and f if a function on S then $f(x)=f(y)$), follow from the axioms for equality.

The development of the theory of equivalence classes (cf. 1.0 iv)) requires the axioms for subtypes and for sets. It turns out here, when translating mathematics in AUT-QE, that Landau goes quite far in considering concepts and statements about those concepts to belong to "logisches Denken".

We had to choose how to describe partial functions in AUT-QE. As an example let us consider the function f on the type rl of the reals, defined for all $x \underline{E} \mathsf{rl}$ for which $x \neq 0$, and mapping x to $1/x$. There are (at least) four reasonable ways to represent f:

i) The range of f may be taken to be rl^*, the "extended type" of reals, containing, apart from the reals, an object und representing "undefined". In this case $\langle 0 \rangle f$ will be (book-equal to) und, and rl may be defined as $\mathsf{OT}(\mathsf{rl}^*, [x, \mathsf{rl}^*](x \neq \mathsf{und}))$.

ii) An arbitrary fixed object in rl , 0 say, may replace und . Then <0>f will be taken to be 0 .

iii) f may be considered as a function on OT(rl,[x,rl]x≠0) , the subtype of the nonzero reals.

iv) f may be represented as a function of two variables: an object x E rl and a proof p E x≠0 . So

f E [x,rl][p,x≠0]rl .

(Then, given an x such that x≠0 , i.e. given an x and a proof p that x≠0 , we can use <p><x>f to represent 1/x .)

It is clear that the representations i) and ii) have much in common. The representations iii) and iv) are also related: in fact, we may construct, by the axioms for subtypes, for given x E rl and p E x≠0 an object out(x,p) E OT(rl,[x,rl]x≠0) . Then, if

f_1 E [x,OT(rl,[x,rl]x≠0)]rl ,

then

[x,rl][p,x≠0]<out(x,p)>f_1 E [x,rl][p,x≠0]rl .

On the other hand, if

f_2 E [x,rl][p,x≠0]rl

then

[x,OT(rl,[x,rl]x≠0)]<OTAx(x)><in(x)>f_2 E [x,OT(rl,[x,rl]x≠0)]rl

(for brevity some obvious subexpressions in the formulas above have been omitted).

After a careful examination of Landau's language, I have decided that the fourth representation is closest to his intention, and have therefore adopted it. However this leads to the following difficulty:

Let, in our example, x E rl and y E rl be given, such that x=y , and suppose we have proofs p E (x≠0) and q E (y≠0) . Now it is not *a priori* clear in AUT-QE (though it is clear to Landau), that the corresponding values <p><x>f and <q><y>f will be equal. That is: it is not guaranteed in the language that the function values for equal arguments will be independent of the proofs p and q .

This property of partial functions, which is called *irrelevance of proofs*, can be proved for all functions which Landau introduces. When discussing arbitrary partial functions however, irrelevance of proofs had to be assumed in some places (cf. gite below). For a further discussion we refer to 4.0.1.

As a consequence of the chosen representation of partial functions, terms may depend on proofs, and therefore certain propositions are meaningful only if others are true. This gives rise to generalized implications (cf. 1.0 vi)) and generalized conjunctions, such as:

"$x > 0 \Rightarrow 1/x > 0$"

and

"$x > 0 \wedge \sqrt{x} \neq 2$" .

Logical connectives of this kind have been formalized in the paragraph "r" in the preliminary AUT-QE text.

The definition-by-cases operator ite (short for "if-then-else", cf. 1.0 vii))) can be defined on the basis of the axioms for individuals. As we have seen (1.0 vii)), Landau admits partial functions in such definitions. For these cases a "generalized" version of the definition-by-cases operator gite (for "generalized if-then-else") is required, which has been defined only for partial functions satisfying the irrelevance of proofs condition.

All set theoretical concepts used by Landau (cf. 1.0 viii)) may be defined starting from our axioms for sets.

The passages in Landau's text which use more or less intuitive reasoning (cf. 1.0 x)) could not very well be translated. In the relevant places straightforward logical proofs were given, which follow Landau's line of thought as closely as possible.

2. TRANSLATION

In this chapter, we discuss the actual translation of Landau's book, the difficulties encountered and the way they were overcome (or evaded). First, in section 2.0, we give an abstract of Landau's book; then, in section 2.1, a general survey is given of the various reasons to deviate occasionally from Landau's text. In the following sections we describe the translation of the chapters 1 to 5 of Landau's book.

2.0. An abstract of Landau's book

i) "Kapitel 1. Natürliche Zahlen".
Peano's axioms for the natural numbers 1,2,3,... are stated.
"+" is defined as the unique operation satisfying $x+1 = x'$ and $x+y' = (x+y)'$. Properties of + (associativity, commutativity) are derived.
Order is defined by $x > y :\equiv \exists u\,[x = y + u]$. It is proved to be a linear order relation and its connections with + are derived. "Satz 27" states that it is a well-ordering.
"." (multiplication) is defined as the unique operation satisfying $x.1 = x$ and $x.y' = x.y + x$. Properties of "." (commutativity, associativity) are derived, and also its connections with + (distributivity) and with order.

ii) "Kapitel 2. Brüche".
Fractions (i.e. positive fractions) are defined as pairs of natural numbers. Equivalence of fractions is defined, and proved to be an equivalence relation.
Order is defined, it is shown to be preserved by equivalence, and to be an order relation. Properties are derived (e.g. it is shown that neither maximal nor minimal fractions exist, and that the set of fractions is dense in itself).
Addition and multiplication are defined, and proved to be consistent with equivalence. Their basic properties and interconnections are derived, and their connections with order are shown. Also subtraction and division are defined.
Rationals (i.e. positive rational numbers) are defined as equivalence classes of fractions. Order, addition and multiplication are carried over to the rationals, and their various properties are proved. Finally the natural numbers are embedded, and the order in the rationals is shown to be archimedean.

iii) "Kapitel 3. Schnitte".

Cuts in the positive rationals are defined.

For these cuts, order, addition (with subtraction), and multiplication (with division), are defined, and again the various properties and interconnections of these concepts are proved.

The rationals are embedded, and the set of rationals is proved to be dense in the set of cuts. Finally the existence of irrational numbers is proved, by introducing $\sqrt{2}$ as an example.

iv) "Kapitel 4. Reelle Zahlen".

The cuts are now identified with the positive real numbers, and to these the real number 0 and the negative reals are adjoined, in such a way that to every positive real there corresponds a unique negative real.

The absolute value of a real number is defined. Order is defined, its properties are derived, and the predicates "rational" and "integral" ("ganz") are defined on the reals.

Now addition and multiplication are defined, and their properties and their connections with each other, with absolute value and with order are derived. In particular the minus operator (associating to each real its additive inverse) is discussed, as well as subtraction and division.

Finally, in the "Dedekindsche Hauptsatz", Dedekind-completeness of the order in the reals is proved.

v) "Kapitel 5. Komplexe Zahlen".

Complex numbers are defined as pairs of reals.

Addition, multiplication, subtraction and division, their properties and interconnections are discussed.

To each complex number is associated its conjugate, and also (following the definition of the square root of a nonnegative real) its modulus (as a real number). The connections of these two concepts with each other and with the previously introduced operations are derived.

For an associative and commutative operator * (which may be interpreted as either + or .), and for an n-tuple of complex numbers f(1),..,f(n), Landau denotes

$$f(1) * f(2) * \ldots * f(n) \text{ by } \sum_{i=1}^{n} f(i) .$$

This concept is defined as the value at n of the unique function g (with domain {1,2,...,n}) for which g(1) = f(1) and g(i') = g(i) * f(i')

for i < n. The properties of $\mathfrak{Z}$ are proved in particular, for any permutation s of {1,2,....n} it is proved that

$$\mathfrak{Z}_{i=1}^{n} f(i) = \mathfrak{Z}_{i=1}^{n} f(s(i)) .$$

The definition of $\mathfrak{Z}$ is extended to n-tuples f(y),f(y+1),..,f(y+n-1) (where y is an integer), and its properties are carried over to this situation. Σ is defined as the specialization of $\mathfrak{Z}$ to the operation +, and Π as its specialization "." (multiplication). Some properties of Σ and Π are proved.

For a complex number x and an integer n, with $x \neq 0$ or $n > 0$, x^n is defined, and its properties and connections with previously defined concepts are discussed.

Finally the reals are embedded in the set of complex numbers, the number i is defined, it is proved that $i^2 = -1$, and that each complex number may be uniquely represented as a+bi with a,b real.

2.1. Deviations from Landau's text

In our translation, deviations from Landau's text appear occasionally. They may be classified as follows:

i) In some cases a direct translation of Landau's proofs seems a bit too complicated. We list three reasons for this.

   a) Sometimes it is due to the structure of AUT-QE which does not quite agree with the proof Landau gives. E.g. in the proof of "Satz 6" Landau applies, for fixed y, induction with respect to x. As x E nat, y E nat is a common context in the translation, it is easier there to apply, for fixed x, induction with respect to y.

   b) Sometimes the reason is that Landau uses a unifying argument. E.g. in the proof of the "Dedekindsche Hauptsatz" there are, at a certain stage, two real numbers Ξ and H, such that $\Xi > 0$ and $\Xi > H$. Here Landau needs a rational number Z, such that $\Xi > Z > H$. Now it has been proved in "Satz 159" that between any two positive reals there is a rational. If $H > 0$ this may be applied immediately. If $H \leq 0$ Landau defines $H_1 = \frac{\Xi}{1+1}$ and again applies "Satz 159", this time with H_1.

   This argument however is complicated, because, to apply "Satz 159", first $0 < H_1 < \Xi$ has to be proved (which Landau fails to do). And it

is superfluous because every Z in the cut Ξ will meet the conditions in this case.

c) In one instance (the proof of "Satz 27"), Landau has given a complex proof, which may be simplified.

In all these cases I have, in the translation, given a proof which follows Landau's line of reasoning. However, in some cases, I have also given shorter alternative proofs. This means that the deviations are optional in these cases.

ii) Some of Landau's "Sätze" really consist of two or three theorems. E.g. "Satz 16: Aus $x \leq y$, $y < z$ oder $x < y$, $y \leq z$ folgt $x < z$". In such cases the theorem has been split up: "Satz 16a: Aus $x \leq y$, $y < z$ folgt $x < z$", "Satz 16b: Aus $x < y$, $y \leq z$ folgt $x < z$".

iii) Very frequently Landau uses without notice a number of more or less trivial corollaries of a theorem he has proved. E.g. besides "Satz 93: $(X + Y) + Z = X + (Y + Z)$" he uses "$X + (Y + Z) = (X + Y) + Z$" without quoting "Satz 79". Sometimes such a practice is explicitly announced, e.g. in the "Vorbemerkung" to "Satz 15", where it is stated that, with any property derived for $<$, the corresponding property for $>$ shall be used. In all such cases the corollaries have been formulated and proved after the theorems.

iv) Following the translation of the definition of a concept, we often added the specialization to this concept of certain general properties. E.g. after the introduction of +, substitutivity of equality was applied: "If $x = y$ then $x + z = y + z$ and $z + x = z + y$. If $x = y$ and $z = u$ then $x + z = y + u$". This was done in order to make later applications easier.

v) In a few proofs of the last three chapters minor changes were made. E.g. in the proof of "Satz 145", where Landau states: "Aus $\xi > \eta$ folgt nach Satz 140 bei passendem υ $\xi = \eta + \upsilon$" but where, by "Definition 35" υ can be defined explicitly by $\upsilon := \xi - \eta$. This has been done in the translation, thus avoiding the superfluous existence elimination. Another deviation occurs in the proof of "Satz 284". Here Landau writes the following chain of equalities:

$$((u+1)-y) + (x-u) = (x+(-u)) + ((u+1) + (-y)) =$$
$$= (x+((-u)+(u+1))) + (-y) = (x+1) - y \ .$$

As in the proof the equality

$$((u+1)-y) + ((x+1) - (u+1)) = (x+1) - y$$

was needed, the following chain of equations was preferred in the translation:

$$((u+1)-y)+((x+1)-(u+1)) = ((x+1)-(u+1))+((u+1)-y) =$$
$$= (((x+1)-(u+1))+(u+1))-y = (x+1)-y \ .$$

vi) As we have seen in 1.0 vii) Landau formulates Peano's fifth axiom in terms of sets, and, when applying it, always represents a predicate as a set. In the translation this extra step has been avoided. The induction axiom is indeed introduced for sets, but then immediately a lemma, called induction , which applies to predicates is proved. This lemma has been used systematically in all proofs by induction.
Also "Satz 27: In jeder nicht leeren Menge natürlicher Zahlen gibt es eine kleinste" has been reworded and proved in terms of predicates and not of "Mengen".

vii) "Intuitive arguments" of Landau were translated in various ways. E.g. "Satz 20: Aus $x + z > y + z$ bzw. $x + z = y + z$ bzw. $x + z < y + z$ folgt $x > y$ bzw. $x = y$ bzw. $x < y$.
Beweis: Folgt aus Satz 19 da die drie Fälle beide Male sich ausschliesszen und alle Möglichkeiten erschöpfen" (where "Satz 19" asserts the inverse implications).
Considering the fact that Landau regards this proof as belonging to "logisches Denken", I have proved in the preliminaries three "logical" theorems to the effect that:
If $A \vee B \vee C$, $\neg(D \wedge E)$, $\neg(E \wedge F)$, $\neg(F \wedge D)$ and $A \Rightarrow D$, $B \Rightarrow E$, $C \Rightarrow F$, then $D \Rightarrow A$, $E \Rightarrow B$ and $F \Rightarrow C$.
These theorems were used in the translation.
A second example: "Satz 17: Aus $x \leq y$, $y \leq z$ folgt $x \leq z$.
Beweis: Mit zwei Gleichheitszeichen in der Voraussetzung klar; sonst durch Satz 16 erledigt" ("Satz 16" is quoted above under ii)). Here the AUT-QE text, when translated back into German, might read:
"Beweis: Es sei $x = y$. Dann ist, wenn $y = z$, auch $x = z$ also $x \leq z$.
Wenn aber $y < z$ so ist $x < z$ nach Satz 16a, also ebenfalls $x \leq z$.
Nehme jetzt an $x < y$. Dann folgt aus Satz 16b $x < z$, also auch in diesem Fall $x \leq z$. Deshalb ist jedenfalls $x \leq z$".
Another argument which is difficult to translate faithfully occurs in "Kapitel 5, § 8" where sums and products are introduced. Landau uses here a symbol which he intends to represent either "+" or ".", and in this way defines "Σ" and "Π" simultaneously. In our translation we de-

fined iteration for arbitrary commutative and associative operators, and consequently our concept and the relevant theorems are essentially stronger than Landau's. This generality is much easier to describe in AUT-QE then a theory which applies only to "+" and ".".

viii) Landau uses metatheorems whenever he embeds one structure into another, to show that the properties proved for the old structure "carry over" to the new. As an example I cite his treatment in chapter 2 of the embedding of the natural numbers into the (positive) rationals.

"Satz 111: Aus $\frac{x}{1} > \frac{y}{1}$ bzw. $\frac{x}{1} \sim \frac{y}{1}$ bzw. $\frac{x}{1} < \frac{y}{1}$
folgt $x > y$ bzw. $x = y$ bzw. $x < y$".

"Definition 25: Eine rationale Zahl heiszt ganz, wenn unter den Brüchen, deren Gesamtheit sie ist, ein Bruch $\frac{x}{1}$ vorkommt".

"Dies x ist nach Satz 111 eindeutig bestimmt, und umgekehrt entspricht jedem x genau eine ganze Zahl".

"Satz 112: $\frac{x}{1} + \frac{y}{1} \sim \frac{x+y}{1}$, $\frac{x}{1} \cdot \frac{y}{1} \sim \frac{x.y}{1}$ ".

"Satz 113: Die ganzen Zahlen genügen den fünf Axiomen der natürlichen Zahlen, wenn die Klasse von $\frac{1}{1}$ an Stelle von 1 genommen wird, und als Nachfolger der Klasse von $\frac{x}{1}$ die Klasse von $\frac{x'}{1}$ angesehen wird".

Landau adds the following comment:

"Da =, >, <, Summe und Produkt (nach Satz 111 und 112) den alten Begriffen entsprechen, haben die ganzen Zahlen alle Eigenschaften die wir in Kapitel 1 für die natürlichen Zahlen bewiesen haben".

It was difficult to translate this text. The translation requires first a careful analysis of the interpretation of Peano's axioms in chapter 1. There are two possibilities:

In the first interpretation, the axioms describe fundamental properties of the given system of naturals (nat, 1, suc), which cannot be proved from more primitive properties, and from which all other properties of the system can be derived. In this conception there is an intention to characterize the structure by the axioms.

In the second interpretation, the axioms are simply assumptions underlying a certain theory. The theorems of the theory are valid in any structure in which these assumptions hold. In this view, no claim is made that the axioms characterize the system.

The difference between these two conceptions can be illustrated by comparing the rôle of the axioms in Euclid's geometry to the rôle of the axioms for groups in group theory.

The interpretation of "Satz 113" and Landau's comment varies according to the interpretation of the Peano axioms. In the first interpretation the "ganzen rationalen Zahlen" form a structure (nat^*, 1^*, suc^*) which "happens to" have the same fundamental properties as the original structure (nat, 1, suc). Hence, by a suitable metatheorem, we see that the reasoning of chapter 1 may be repeated for this new structure, extending it to (nat^*, 1^*, suc^*, $+^*$, $.^*$, $<^*$) and proving the various properties of this extended system.

In the second interpretation "Satz 113" just proves that the structure (nat^*, 1^*, suc^*) satisfies the assumptions. After this the theory of chapter 1 can be applied immediately.

However there is a further problem (under either interpretation): addition on nat^* defined according to the method of chapter 1 is *not* (definitionally) the same thing as the restriction (to nat^*) of the addition on the rationals and these two functions must still be *proved* to be (extensionally) equal. Similar remarks can be made about multiplication and order.

It follows that the relevant text cannot be rendered directly in AUT-QE under either interpretation of Peano's axioms. There is, therefore, no technical reason to prefer one of these interpretations to the other.

Landau's ideas on the rôle of the axioms are not quite clear from his text. We cite some of his statements:

- In his "Vorwort für den Kenner" he mentions certain laws on the reals which can be "als Axiome postuliert".
- He thinks it right, that the student should learn "auf welchen als Axiomen angenommenen Grundtatsachen sich lückenlos die Analysis aufbaut".
- Moreover: "In dieser (Vorlesung) gelange ich, von den Peanoschen Axiomen der natürlichen Zahlen ausgehend, bis zur Theorie der reellen Zahlen".
- In chapter 1: "Wir nehmen als gegeben an:
  Eine Menge, d.h. Gesamtheit, von Dingen, natürliche Zahlen genannt, mit den nachher aufzuzählenden Eigenschaften, Axiome genannt".
- "Von der Menge der natürlichen Zahlen nehmen wir nun an, dasz sie die Eigenschaften hat...".
- A relevant passage is also "Satz 113" quoted above.
- Landau never mentions "a system of naturals", like in group theory one would discuss "a group", but always "die natürlichen Zahlen".

Most of the sentences quoted above point to the second interpretation, some of them however could be interpreted better or equally well in the first way.

Now, as neither technical reasons nor Landau's text indicated definitely how Peano's axioms should be interpreted, I decided to interpret them as postulates (PN-lines) rather then assumptions (EB-lines) because it suited my own conception of the naturals. Moreover this interpretation reduces the context and thereby simplifies verification.

The meta-reasoning sketched above has been treated as follows. After the proof of "Satz 113" the proofs of "Satz 1" and "Satz 4" (where addition is introduced) were copied for the "ganzen Zahlen". However addition on the "ganzen Zahlen" has been defined as the restriction of addition on the rationals. Then a number of theorems from "Kapitel 1" where proved using "Satz 112". Order and multiplication were treated in a similar way. These texts have been inserted as a matter of prestige because we claimed that we were able to say everything Landau says. The insertions were never used however (cf. ix) below).

In "Kapitel 3, § 5" and "Kapitel 5, § 10" similar arguments occur, when the rationals are embedded in the reals, and the reals in the complex numbers. These arguments were "translated" just by constructing the relevant isomorphisms. This suffices for all applications.

ix) A consequence of the difficulties described in viii) is a divergence between the translation and Landau's book with respect to the use of natural numbers in the chapters 3, 4 and 5. After his comment (following "Satz 113") that the "ganze Zahlen" have the same properties as the "natürliche Zahlen" Landau continues:

"Daher werfen wir die natürlichen Zahlen weg, ersetzen sie durch die entsprechenden ganzen Zahlen, und haben fortan (da auch die Brüche überflüssig werden) in bezug auf das Bisherige nur von rationalen Zahlen zu reden".

In the translation I have not followed this course, because, as pointed out, it would have been a cumbersome task to prove the properties of the "natürliche Zahlen" for the "ganze Zahlen", and also because it would have been inevitable to repeat this procedure with every further extension of the number system. Therefore I have stuck to the "natürliche Zahlen" throughout the translation.

x) Another important deviation of Landau's text was caused by "Definition 43: Wir erschaffen eine neue, von den positiven Zahlen verschiedene Zahl 0. Wir erschaffen ferner Zahlen die von den positiven und 0 verschieden sind, negative genannt, derart, dasz wir jedem ξ (d.h. jeder positiven Zahl) eine negative Zahl zuordnen, die wir $-\xi$ nennen".

I doubt wether this creative act may be called a "definition". Landau considers it a part of "logisches Denken" to form, given sets (or types) α and β, the cartesian product $\alpha \times \beta$, as is clear from chapter 2. It might be also considered "logical" to form the disjoint union $\alpha \oplus \beta$. But Landau does not mention this, he just "creates" 0 and the negative numbers from nothing.

Moreover I do not see a formal difference between the assertion "1 ist eine natürliche Zahl" (which Landau calls an axiom) and the assertion "0 ist eine reelle Zahl" (which he calls a definition). Neither do I see a formal difference between "$x' \neq 1$" and "$-\xi \neq 0$". In my opinion the limits of "logisches Denken" are exceeded here.

In agreement with this criticism I have translated this "definition" by introducing a number of primitive concepts and axioms (PN-lines). The type of real numbers `rl` is a primitive type. To any cut ξ real numbers `p(`ξ`)` and `n(`ξ`)` are associated. `0` is a primitive real number. Next there are axioms to the effect that the functions `[x,cut]p(x)` and `[x,cut]n(x)` are injective. Now `x E rl` has the property `pos` (or `neg`) if it is in the range of the first (or the second) of these functions. Then there are axioms stating that, for `x E rl` , `pos(x)` , `neg(x)` and `x=0` are mutually exclusive, and that each `x E rl` has one of these properties. (In fact Landau does not state the latter axiom explicitly.) Starting from these axioms "Kapitel 4" was translated.

However, as I thought it unsatisfactory to develop the theory of real and complex numbers using more than Peano's axioms alone, I have added an alternative AUT-QE version of chapter 4, called chapter 4a, where the real numbers are defined as equivalence classes of pairs of cuts, and where all theorems of Landau's "Kapitel 4" are proved for these alternative reals. The AUT-QE translation of chapter 5 has been checked relative to the AUT-QE book consisting of the chapters 1, 2, 3 and 4a.

2.2. The translation of "Kapitel 1"

§ 1. Equality was introduced in the preliminaries (cf. 1.2 ii) and 1.3). nat is introduced as a primitive type, the Peano axioms as PN-lines (cf. 2.1 viii)). Induction is formulated in terms of sets, but immediately a lemma on induction, which applies to predicates is proved. This lemma is used in the sequel (cf. 2.1 vi)).

§ 2. "Satz 4: Auf genau eine Art läszt sich jedem Zahlenpaar x,y eine natürliche Zahl, x + y genannt, so zuordnen, dasz..." has been translated the way it is proved by Landau, viz. "for each x E nat there exists a unique function f E [t,nat]nat such that...". (In fact this theorem might have been proved without using extensional equality of functions.)

After the proof of "Satz 4" we have in the translation 11 corollaries and lemma's (cf. 2.1 iii) and 2.1 iv)). To some of these Landau refers explicitly (in the proof of "Satz 6": "nach dem Konstruktion beim Beweise des Satzes 4") but more often they are used implicitly (e.g. in the proofs of "Satz 9" and "Satz 24").

§ 3. Landau's "Definition 2: Ist x = y + u so ist x > y" is a bit loose and requires of course a better formalization. His proof of "Satz 27" is not very well organized, and uses indirect reasoning twice. After the translation of this proof in AUT-QE (36 lines, 458 identifier occurrences) a more straightforward proof was given (reducing the length to 23 lines, 264 identifier occurrences). This alternative proof, translated back into German (with "Mengen" instead of predicates, cf. 2.1 vi)), might read as follows:
"Satz 27: In jeder nicht leeren Menge natürlicher Zahlen gibt es eine kleinste".
Beweis: N sei die gegebene Menge, M die Menge der x die ≤ jeder Zahl aus N sind. Nehme an es gibt in N keine kleinste.

1 gehört zu M nach Satz 24.

Ist x zu M gehörig so ist x ≤ jeder Zahl aus N. x gehört nicht zu N, den sonnst wäre x kleinste Zahl aus N. Nach Satz 25 ist also jeder Zahl aus N ≥ x + 1, und daher gehört x + 1 zu M.

M enthält somit jede natürliche Zahl.

Wenn aber y zu N gehört, so gehört, wegen y + 1 > y, y + 1 nicht zu M, gegen das Obige.

N enthält also eine kleinste Zahl".
(The German proofs do not differ too much in length: they contain 139 resp. 116 words.)

§ 4. The theorems on multiplication and their proofs are very similar to those on addition. The remarks made above concerning the translation of § 2 apply here too.

After the translation of "Kapitel 1", in our AUT-QE text, for each x E nat , the type 1to(x) of the natural numbers ≤ x is defined. Then, for an arbitrary type S , the type pair1type(S) is defined to be [t,1to(2)]S . It represents the type of pairs (a,b) with a E S , b E S . Its various properties are then derived (cf. 1.2 v)).

2.3. The translation of "Kapitel 2"

§ 1. Landau defines fractions as ordered pairs. However he does not use variables for pairs, but indicates them by their components: " $\frac{x_1}{x_2}$, $\frac{y_1}{y_2}$ " etc. In the translation x is a variable for fractions, with numerator num(x) and denominator den(x) . And to x1 E nat , x2 E nat is associated the fraction fr(x1,x2) .

§ 5. The rationals are defined as equivalence classes of fractions. The subsequent proofs have all the same structure: in the equivalence classes representatives are chosen, and the theorems proved for these representatives are carried over to their classes. (Landau's description of this course of reasoning is rather sketchy, e.g.: "Satz 81: .... Beweis: Satz 41".)

In order to translate this practice, four lemmas were proved, covering the cases where 1, 2, 3 or 4 rationals are involved, and which are used throughout the translation of § 5.

After the proof of "Satz 112" it is proved (as an extra theorem) that for two "ganzen Zahlen" x and y, such that x > y, the difference x - y is also "ganz". Landau uses this (without proof) in his proofs of "Satz 162" and "Satz 285".

The translation of "Satz 111", "Definition 25", "Satz 112" and "Satz 113", with the ensuing text on "throwing away" the naturals, has been extensively discussed already in 2.1 viii).

2.4. The translation of "Kapitel 3"

§ 1. The definition of the concept "Schnitt" did not give rise to difficulties. The type cut is defined as the type of those sets of rationals which are cuts. Now, in this definition, there are three properties of cuts ξ which involve existential quantification:

i) ξ is not empty: $\exists x\ [x \in \xi]$.

ii) the complement of ξ is not empty: $\exists x\ [x \notin \xi]$.

iii) ξ contains no maximal element: if $x \in \xi$ then $\exists y\ [y \in \xi \wedge y > x]$.

Therefore, if ξ is a cut, then there are three ways to apply existence elimination. Three lemmas to that effect (which Landau uses without notice) are stated and proved in the AUT-QE text immediately after the introduction of the concept cut .

Also in other paragraphs in this chapter, when existential quantification was used in defining relations ($>$ in § 2) or objects ($\xi + \eta$ in § 3, $\xi.\eta$ in § 4), a corresponding existence elimination rule was stated and proved as a lemma immediately afterwards.

§ 3. "Satz 132. Bei jedem Schnitt gibt es, wenn A gegeben ist, eine Unterzahl X und eine Oberzahl U mit U - X = A" is an example of the use of "generalized" logic as described in 1.3. In fact, as U and X are positive rationals, the term U - X is only defined if U > X. That this is the case is a consequence of the assumption that U and X are "Oberzahl" resp. "Unterzahl" of the same cut ξ (i.e. $U \notin \xi$ and $X \in \xi$).

In the proof of "Satz 140" there is a reference to the "Anfang des Beweises des Satzes 134". In Landau's Satz-Beweis style this is slightly unorthodox. In AUT-QE there is no such objection. The translation of this reference is given in a single AUT-QE line referring to a line in the proof of "Satz 140".

§ 4. Preceding the proof of "Satz 141" there is in the AUT-QE translation a lemma stating that for rationals X and Z we have $\frac{1}{X} . Z = \frac{Z}{X}$. This is used without proof by Landau in the proofs of "Satz 141" and "Satz 145".

§ 5. Embedding the (positive) rationals in the (positive) reals, (i.e. in the type cut), gives rise to difficulties as described in 2.1 viii).

Finally, it is proved in the translation (as a corollary of "Satz 112") that, for cuts ξ and η which are (embedded) naturals, $\xi + \eta$, $x.\eta$ and (if $\xi > \eta$) $\xi - \eta$ are (embedded) naturals too. These results are used in "Kapitel 5, § 8".

2.5. The translation of "Kapitel 4"

§ 1. The first definition of this chapter and its translation have been discussed in 2.1 x): Contrary to Landau's intentions, in the translation the cuts from chapter 3 are not identified with positive reals. This is because we want to collect the reals in a single type rl , and because

types in AUT-QE are unique. (Accordingly there are in AUT-QE no facilities for extending types; we always have to use embeddings instead.) Some proofs in this chapter are complicated by this distinction between cuts and positive reals.

§ 2. The very complicated definitions by cases in this chapter were occasionally slightly modified. E.g.:
"Definition 44:

$$|\Xi| = \begin{cases} \xi & \text{wenn } \Xi = \xi \\ 0 & \text{wenn } \Xi = 0 \\ \xi & \text{wenn } \Xi = -\xi". \end{cases}$$

was translated as

$$|\Xi| = \begin{cases} p(\xi) & \text{if } \Xi = n(\xi) \\ \Xi & \text{otherwise} \end{cases}$$

(here $p(\xi)$ and $n(\xi)$ denote the positive and negative reals associated with the cut ξ).

§ 3. The translation of "Definition 52" (quoted in 1.0 vii)) was tiresome (it took about 180 AUT-QE lines). Equally tedious to translate were the proofs of the theorems following this definition ("Satz 175", "Satz 180". "Satz 185"). In the proof of "Satz 182" it is left to the reader to check the theorem in a number of cases. This task could not be left to a non-human reader without further instructions.

In the proof of "Satz 185" the order in which the 11 different cases are treated has been altered in the translation. The essence of the proof has not been changed, however.

§ 4. The definition of multiplication, where 6 cases are discerned, gave rise to similar difficulties as the definition of addition (it took about 110 AUT-QE lines).

I had some doubts how to interpret
"Satz 196: Ist $\Xi \neq 0$, $H \neq 0$, so ist

$$\Xi.H = |\Xi|.|H| \text{ bzw. } \Xi.H = -(|\Xi|.|H|)$$

je nachdem keine oder zwei, bzw. genau eine der Zahlen Ξ,H negativ sind".
At first sight this seems to mean:

a) If Ξ and H are not negative then $\Xi.H = |\Xi|.|H|$.
b) If Ξ and H are negative then $\Xi.H = |\Xi|.|H|$.
c) If Ξ not negative, H negative then $\Xi.H = -(|\Xi|.|H|)$.
d) If Ξ negative, H not-negative then $\Xi.H = -(|\Xi|.|H|)$.

However, if this meaning is intended the condition $\Xi \neq 0$, $H \neq 0$ is superfluous. Therefore, possibly, the statement is meant to include also

e) If $\Xi.H = |\Xi|.|H|$ then neither or both of Ξ and H are negative.

f) If $\Xi.H = -(|\Xi|.|H|)$ then Ξ is negative and H is not, or H is negative and Ξ is not.

Landau's proof ("Beweis: Definition 55") does not give a clue, and in later references to the theorem he only uses a), b), c) or d). Nevertheless I have formalized proofs of e) and f) in the translation.

"Satz 194" and "Satz 199" have complicated proofs by cases, which were not easy to formalize.

§ 5. The "Vorbemerkung" to "Satz 205" requires two proofs. Some lemmas are needed for the proof of the "Hauptsatz" itself, e.g. it is used that $\frac{1}{\Xi} . H = \frac{H}{\Xi}$ (cf. 2.4). No special difficulties arose in proving this important theorem.

2.6. The alternative version of chapter 4

Our motivation to write another version of chapter 4, called chapter 4a, was discussed in 2.1 x). In this chapter the theorems of chapter 4 are proved for reals which are defined in a way different from Landau's. Also the order in which these theorems appear differs from Landau's order.

At the end of this chapter the square root of a nonnegative real is defined using "Satz 161", and its properties are derived. (This has been done by Landau in "Kapitel 5, § 7").

The lengths of the AUT-QE texts of chapter 4 and chapter 4a are about equal.

2.7. The translation of "Kapitel 5"

The actual translation of this chapter is preceded by a number of lemmas. Some of these give properties of division on the reals, implicitly used by Landau in the sequel. Further there are lemmas describing the shift of a segment of integers $y, y+1, y+2, \ldots, x$ to an initial segment of the naturals $1, 2, \ldots, (x+1) - y$, which serve the translation of § 8.

The translation of the first seven paragraphs of this chapter was straightforward. Preceding the proof of "Satz 221" some lemmas appear, describing, for a complex number x, the properties of $\mathrm{Re}(x)^2 + \mathrm{Im}(x)^2$. These properties are used by Landau without notice in the proofs of "Satz 221"

and "Satz 229" and in the definition of $|x|$ ("Definition 66"). (In my opinion, at least a remark should have been made in this definition, to the effect that $Re(x)^2 + Im(x)^2 \geq 0$ for complex x).

§ 8. The translation of this paragraph was difficult. Landau discusses x-tuples of complex numbers in order to define their sums and products. He introduces the concept of an x-tuple as follows: "Es sei f(n) für $n \leq x$ definiert", and explains this later on: "Unter "definiert" verstehe ich "als komplexe Zahl definiert". After proving some theorems he extends the concept to x-tuples indexed by segments of (possibly negative) integers: "In Definition 70 und Satz 284 bis Satz 286 bezeichnen ausnahmsweise lateinische Buchstaben ganze (nicht notwendig positive) Zahlen.
Es sei $y \leq x$, f(n) für $y \leq n \leq x$ definiert....".

There are (at least) three natural ways to represent in AUT-QE the concept of x-tuple indexed by an initial segment of the naturals:

i) f might be considered as a function from the type `nat` to the type `CX` of complex numbers, of which only the first x values are taken into account. If we take this attitude it should be proved that if f and g coincide for $n \leq x$ then their sums (and products) up to x are equal.

ii) f might be represented as a function of type `[t,nat][u,t≤x]cx` , i.e. as a partial function like those discussed in 1.3.

iii) f might be considered as a function having as its domain the type `1to(x)` , the subtype of those naturals which are $\leq x$.

All these possibilities have certain advantages. The first one is probably the easiest one, the second is in better harmony with the rest of our AUT-QE translation, the third maybe corresponds better with Landau's intentions.

The third formalization was finally chosen, but caused quite some trouble because (on account of the unicity of types) numbers of type `1to(x)` do not have also type `1to(x+1)` .

As to the formalization of x-tuples indexed by segments of the integers, there was the extra difficulty that the predicate "ganze Zahl" on the reals is not thoroughly discussed by Landau. E.g. he does not prove that the integers are closed under addition and subtraction, though he uses this in the text.

For this reason it seemed inappropriate to define the type of integers as a subtype of the reals, and to define f as a (partial) function on this type in one of the ways discussed above.

Therefore we defined f, for fixed integers x and y, as a function of type [t,real][u,int(t)][u,y≤t≤x]cx , i.e. as a partial function on the reals (rather like [t,nat][u,t≤x]cx , see ii) above).

With this formalization of x-tuples (resp. ((x+1)-y)-tuples) the translation of § 8 turned out to be laborious. Many rather meaningless embedding and lifting functions appear in the proofs. In particular the proof of "Satz 283" where it is shown that sums (products) are invariant under permutations of their terms (factors) turned out to be long and tedious. (It should be remarked that Landau's proof is long too: 4 pages, 87 lines of German text, while the translation needs 365 lines of AUT-QE text.)

The last two paragraphs did not present difficulties in translating.

3. VERIFICATION

In this chapter the verification of the AUT-QE text is described. Some features of the program and the possibility of excerpting are discussed.

3.0. Verification of the text

The verification of the AUT-QE translation of Landau's book was executed on the Burroughs B6700 computer at the Technological University of Eindhoven. The last page of the book was checked in September 1975. The whole book was checked in a final run on October 18, 1975. The verifying program was conceived by N.G. de Bruijn and implemented by I. Zandleven. For a description of this program we refer to [Z1]. Zandleven also provided the program with input and output facilities, and extended it with a conversational mode for on-line checking and correcting of texts.

The verification took place in three stages:

i) First the AUT-QE text was fed into the system on a teleprinter. At this stage the main syntactical structure of the text was analyzed. It was checked, for example, that the format of the lines was as it should be, that the bracketing of the expressions was correct, and that no unknown identifiers occurred.

ii) Secondly the AUT-QE text was coded. At this stage the correct use of the context structure, the validity of variables, the correct use of the *shorthand facility* [vD, 2.15] and of the paragraph reference system (cf. appendix 2), were checked.

iii) Finally the text was checked with respect to all clauses of the language definition. At this stage the *degrees* [vD, 2.3] and types of expressions were calculated, and the correctness of application expressions and constant expressions was checked. Vital for this is the verification of the definitional equality of certain types (cf. [vD, 2.10], [Z1]).

Runs of the stages ii) and iii) generally claimed much of the computers (virtual) memory capacity (over 6000K bytes was needed for the program together with the coded text). In order to avoid congestion in the multiprogramming system it was therefore necessary to have the program executed at night (and off-line). As AUTOMATH texts are checked relative to correct books, a mechanical provisional debugging device for off-line checking was implemented, by which lines which were found incorrect could be tentatively repaired.

E.g., when the *middle part* [vD, 2.13.1] of a line was found incorrect, the debugging device changed it temporarily into PN, thus turning an abbreviation line into a PN-line. The line so "corrected" was then again checked, and, if it was found correct, the lines following could then be checked relative to the "corrected" book. By this device it was not necessary to stop the checking immediately after the first error had been found.

Another feature of the verifying program was added because of the fact that proving expressions to be incorrect (especially proving expressions to be not definitionally equal) is often more difficult and more time-consuming then proving correctness. Therefore during off-line runs a parameter in the program (viz. the number of *decision points*, to be explained in 3.1) has been limited, and lines were considered provisionally incorrect when this limit was exceeded.

When the later chapters were checked, we reduced the demands on the computers memory capacity by abridging the book relative to which the text was checked, in the following way: in the chapters which had already been found correct, the proofs of theorems and lemmas were omitted, and the final lines of these proofs (where the theorems and lemmas are asserted) were changed into PN-lines. Each time a chapter was completely checked (relative to the book so abridged) it was abridged in its turn.

Texts which are correct relative to the abridged book will be correct with respect to the unabridged book too. On the other hand, as in classical mathematics there is no reference to proofs but only to assertions, it is unlikely that texts which are correct relative to the unabridged book will be rejected relative to the abridged book. In actual fact this did not occur.

When a chapter, after several off-line runs of the program, was found to be "nearly correct", the final verification of that chapter took place on-line. In such an on-line run the remaining errors could be immediately corrected. Moreover correct lines could be verified, which had been provisionally rejected because the number of decision points during verification in off-line runs had exceeded the chosen limit. The verification of such complicated lines could be shortened by directing (in conversational mode) the strategy for establishing definitional equality.

After all chapters were verified in this way, the integral AUT-QE text (complete and unabridged) was checked during a final on-line run, which took 2 hours (real time). Of this time 42 min was spent on verification (not including the time needed for coding).

In a table we list some data on this final run, concerning verification time, number of performed reductions and memory occupied

	preliminary text	chapter 1	chapter 2	chapter 3	chapter 4a	chapter 5	chapter 4	complete text
verification time in seconds	107.3	143.1	301.2	342.4	405.7	813.1	406.9	2519.7
α-reductions	631	752	1077	1455	1644	3393	1533	10485
β-reductions	564	832	460	466	414	2749	529	6014
δ-reductions	596	1111	1318	1873	2724	9290	3151	20063
η-reductions	2	-	-	-	-	-	-	2
nr. of lines	1068	886	1603	2181	2779	2690	2226	13433
nr. of expressions	9388	12155	25792	30327	42067	60450	34959	215138

Since one coded expression occupies about 30 bytes (mainly used for references to subexpressions), the total memory required for the coded book is about 6500 K bytes (= 52000 K bits).

3.1. Controlling the strategy of the program

In order to establish definitional equality of two expressions, the verification system tries to find another expression to which both reduce. The choice of efficient reduction steps for this purpose is a matter of strategy ([vD, 6.4.1]). The programmed strategy is described in [Z1].

Under this strategy it is possible that intermediate results are obtained which strongly suggest a negative answer to the question of definitional equality, without definitely settling it. Suppose, for example, that $a(p)=a(q)$ has to be established. The programs strategy is to ascertain that the constants a and a are identical and to verify whether $p=q$. If this is not the case, there is a strong suggestion that $a(p)$ and $a(q)$ are not definitionally equal either, but this is yet uncertain. For example, they are definitionally equal relative to the book

* n :=	PN	E type
* p :=	PN	E n
* q :=	PN	E n
* x :=	—	E n
x * a :=	p	E n

It is a matter of strategy how to proceed in such cases. We may either apply δ-reduction (in which case the issue will be eventually settled) or we may try to continue the verification process without using $a(p)=a(q)$.

Such a situation is called a *decision point*. In on-line runs the verification may be controlled here by the human operator. (Actually, in the situation sketched above, information will be supplied, and the question will appear whether δ-reduction should be tried.) In off-line runs δ-reduction will be applied in order to get a definite answer to the question, and it will be checked that the total number of decision points passed during the checking of a line does not exceed the chosen limit (cf. 3.0).

3.2. A new verifying program

In appendix 5 two shortcomings of the original verifying program are indicated. Due to these shortcomings there was in 1975 no complete (mechanically sustained) certainty that the AUT-QE text was correct.

Meanwhile on entirely new program has been implemented by I. Zandleven and A. Kornaat. In this program clash of variables is impossible because the coding system uses a nameless representation of variables (cf. [dB2]). Moreover the program is constructed in such a way that its claims on the computer's memory capacity can be kept at an acceptable level, thus avoiding the difficulties with on-line runs described in 3.0.

In April 1977 the whole Landau book was checked by this program, and thus the correctness of the AUT-QE text is now mechanically established. We present some data on this run:

	preliminary text	chapter 1	chapter 2	chapter 3	chapter 4a	chapter 5	chapter 4	complete text
verification time in seconds	107.3	143.1	301.2	342.4	405.7	813.1	406.9	2519.7
α-reductions	631	752	1077	1455	1644	3393	1533	10485
β-reductions	564	832	460	466	414	2749	529	6014
δ-reductions	596	1111	1318	1873	2724	9290	3151	20063
η-reductions	2	-	-	-	-	-	-	2
no. of lines	1068	886	1603	2181	2779	2690	2226	13433
no. of expressions	9388	12155	25792	30327	42067	60450	34959	215138

	preliminary text	chapter 1	chapter 2	chapter 3	chapter 4a	chapter 5	chapter 4	complete text
verification time in seconds	196.4	237.5	538.0	619.9	892.7	1631.6	761.0	4877.1
β-reductions	522	728	466	548	462	3636	560	6924
δ-reductions	619	1114	1321	1873	2793	11219	3211	22142
η-reductions	2	-	-	-	-	-	-	2

The differences in verification time are probably due to the extra effort which is put into the organisation of the memory. There are no α-reductions since variables are nameless in the coding system of this program. The slight differences in the numbers of β- and δ-reductions are caused by differences in the verification strategies of the two programs.

3.3. Excerpting

Let B be an AUT-QE book, i.e. a finite sequence of lines. A *subbook* of B is a subsequence of this sequence. A program, called excerpt, is available which, given a correct book B and a line ℓ of B, produces the minimal correct subbook of B containing ℓ. (It is possible to have the line provisionally changed into a PN-line before the subbook is produced.)

This program will display all concepts relevant to the definition of a given concept, and all theorems (with their proofs) used (explicitly or implicitly) in the proof of a given theorem. (If the line is first changed into a PN-line, the program will just give the assumptions under which the theorem holds, and the concepts necessary to understand its contents.)

As an example, we give in appendix 4 an excerpted text for "Satz 27".

4. CONCLUSIONS

In this chapter we discuss some possibilities to represent logic in AUTOMATH. We indicate some desirable extensions of AUT-68 and AUT-QE, and we discuss some aspects (positive as well as negative) of our translation.

4.0. Formalization of logic in AUTOMATH

In this section we shall describe various possibilities to represent systems of natural deduction in AUT-68 ([vD, 2]), in AUT-QE and in some closely related languages. First we discuss two main decisions which have to be made when choosing between these possibilities. Then we indicate explicitly two possibilities to represent logic.

4.0.0. First order v. higher order

In most AUTOMATH languages there are certain restrictions on abstraction. E.g. in AUT-68 as well as in AUT-QE correct abstraction expressions have the form $[x,\alpha]A$ where α is a 2-expression (and hence x, having type α, is a 3-variable, i.e. a variable which is a 3-expression).

Such restrictions allow a faithful representation of first order logic (in the sense of excluding higher order formulas and inferences). In AUT-68 as well as in AUT-QE this can be done by representing propositions and predicates as 2-expressions (as described in [vD, 3]). Then proposition variables and (in AUT-QE) predicate variables will be 2-variables and abstraction (or quantification) with respect to such variables is impossible in the language. If, in such a setting, we want to discern between proposition variables and predicate variables then it is necessary to have abstraction expressions of degree 1 in the language, i.e. to use AUT-QE (and not AUT-68).

In order to represent higher order logic we should require the possibility of abstraction with respect to proposition and predicate variables. Therefore, if we stick to the abstraction restrictions of AUT-68 or AUT-QE, we should represent propositions and predicates by 3-expressions. We may proceed in two ways:

i) we can associate to each proposition a (primitive) type (which we will call the *assertion type* of the proposition). Objects of this type will be considered as proofs of the proposition. In other words: we consider the proposition as asserted iff its assertion type contains some object. This possibility will be elaborated in 4.0.2.

ii) we can extend the language to a new language, called AUT-4, by admitting 4-*expressions* (having 3-expressions as their types (cf. [vD, 2.3]). Then a proposition (represented by a 3-expression) might be considered as asserted if it contains something (some 4-expression). Thus propositions act as their own assertion types, and the representation of logic is just as described in [vD, 3.2], but for a shift with respect to degrees.

4.0.1. Relevance of proofs v. irrelevance of proofs

In all representations of logic in AUTOMATH languages which have been developed so far, proofs (i.e. names of proofs) appear in the language ([vD, 3], [dB], [dV]). In this respect these representations reflect a constructive conception of logic, in which proofs and objects are treated similarly.

In a classical conception of logic, proofs are discussed in the metalanguage only. As a consequence it is impossible in such a conception to discern (in the language) between different proofs of one proposition. This point of view can be roughly represented in AUTOMATH by proclaiming, for any given proposition a, all proofs of a to be equal. This deprives these proofs of their identity, their names should be considered only as references to the place in the book where the proposition is asserted. This possibility has been first suggested by de Bruijn.

If, in a representation of logic in AUTOMATH, such an attitude is adopted, we shall say that this representation satisfies *irrelevance of proofs*. (Cf. [Z], and also 1.3). How this irrelevance of proofs is implemented (i.e. in which sense proofs are considered "equal") will depend both on the language and on the way logic is represented in it (cf. 4.0.3 i) and ii)).

4.0.2. A representation of logic in AUT-68

A higher order system of natural deduction can be formalized in AUT-68 as follows.

A type of propositions is introduced as a primitive type:

$*$ PROP := PN E type

and to each proposition A its assertion type $\vdash(A)$ is associated:

$*$ A := — E PROP

A $*\vdash$:= PN E type

(In earlier publications on AUT-68, bool and TRUE were used instead of PROP and ⊢). If S is a type, an object P E [x,S]PROP has to be interpreted as a predicate. Objects of type [x,S]⊢(<x>P) must then be interpreted as proving that P holds for every x E S . So we want to introduce the proposition ∀(S,P) which has the property that its assertion type contains elements iff the type [x,S]⊢(<x>P) contains elements. This is expressed in the following lines:

* S :=		—	E type
S * P :=		—	E [x,S]PROP
P * ∀ :=		PN	E PROP
P * a :=		—	E S
a * u :=		—	E ⊢(∀(S,P))
u * ∀e :=		PN	E ⊢(<a>P)
P * u :=		—	E [x,S]⊢(<x>P)
u * ∀i :=		PN	E ⊢(∀(S,P))

Starting from these primitive concepts and axioms, higher order logic can be developed. An indication of how this can be done is given in appendix 6, where the first three theorems from Landau's book are derived on the basis of the logic so developed.

This logic represents a constructive system of natural deduction. Axioms could be added for extensional equality of functions and *extensional equality of propositions* (i.e. if a ↔ b then a = b).

Classical logic could be represented this way by adding axioms for irrelevance of proofs:

* A	:=	—	E PROP
A * u	:=	—	E ⊢(A)
u * v	:=	—	E ⊢(A)
v * irr.pr.	:=	PN	E IS(⊢(A),u,v)

and for the double negation law:

A * u	:=	—	E ⊢(¬(¬(A)))
u * d.n.l.	:=	PN	E ⊢(A)

4.0.3. A representation of logic in AUT-QE

How logic can be represented in AUT-QE is described in [vD, 3]. This system, a first order system of natural deduction, has been used in our translation. An indication of the development of logic in it can be found in the excerpted text in appendix 7, which covers the proofs of the first three theorems of Landau's book and the logic used in these proofs.

The system is a bit ambivalent, because it is classical (containing the double negation law as an axiom) but does not satisfy irrelevance of proofs. There are two obvious ways to implement irrelevance of proofs:

i) by adding an axiom:

* A	:=	—	E prop
A * S	:=	—	E type
S * t	:=	—	E [x,A]S
t * u	:=	—	E A
u * v	:=	—	E A
v * irr.pr.	:=	PN	E IS(S,<u>t,<v>t)

That is: if to every proof of A an object of type S is associated, then this object is independent of the nature of the proof. It has been indicated by J. Zucker that this axiom implies irrelevance of proofs in partial functions as mentioned in 1.3:

* S	:=	—	E type
S * T	:=	—	E type
T * P	:=	—	E [x,S]prop
P * f	:=	—	E [x,S][y,<x>P]T
f * a	:=	—	E S
a * b	:=	—	E S
b * u	:=	—	E IS(S,a,b)
u * v	:=	—	E <a>P
v * w	:=	—	E <b>P
w * Q	:=	[x,S][y,<x>P]IS(T,<v><a>f,<y><x>f)	E [x,S]prop
w * ℓ_1	:=	[y,<a>P]irr.pr.(<a>P,T,<a>f,v,y)	E <a>Q
w * ℓ_2	:=	ISP(S,Q,a,b,u,ℓ_1)	E <b>Q
w * ℓ_3	:=	<w>ℓ_2	E IS(T,<v><a>f,<w><b>f)

ii) by extending, in the language, the relation of definitional equality, in such a way that two 3p-expressions (cf. 1.1) are definitionally equal iff their types are definitionally equal. This has been done in the language AUT-Π (cf. [Z]), but could be done in a variant of AUT-QE as well.

If we want to formalize intuitionistic logic in AUT-QE we should have the absurdity rule (i.e. contradiction implies any proposition) instead of the double negation law. The logical connectives (apart from implication) and the existential quantifier could be added as primitive constants, and their elimination- and introduction rules as axioms.

4.1. The language

In this section we discuss some features of AUTOMATH languages, and the value of these features for the formalization of mathematics.

4.1.0. AUT-SYNT

Consider the following AUT-QE text, representing the introduction rule for conjunction:

```
    * a    :=        —          E prop
a * b      :=        —          E prop
b * u      :=        —          E a
u * v      :=        —          E b
v * andi   :=      -----        E and(a,b)
```

(where the dots indicate some proof which is irrelevant for the present discussion). We will call the variables a,b,u,V the *parameters* of andi . If we want to apply this rule for propositions A and B, we need two proofs p and q of the propositions, thus getting the proof andi(A,B,p,q) E and(A,B).

Suppose we are given the proof p, then we can compute mechanically its type (Cf., [vD, 6.4.2.3]) which is (definitionally equal to) the proposition A it proves. A similar observation holds for q and B. Hence we could say that the expression andi(A,B,p,q) contains redundant information. If the "*mechanical type*" *function* CAT ([vD, 6.4.2.3]) were incorporated in the language, we could write, instead of the expression above, andi(CAT(p),CAT(q), p,q), which only contains p and q. We will call the parameters u and v (for which p and q are substituted) the *essential parameters* of andi , while a and b (for which the redundant expressions A and B are substituted)

are called *redundant parameters*. There are many other examples of expressions with redundant parameters.

It is worth while to extend the language in such a way that redundant parameters can be avoided, because the expressions which have to be substituted for them might be long. A system of extensions of this kind has been proposed by I. Zandleven. It is called AUT-SYNT since it admits *syntactic variables* for expressions. Thus we have the languages AUT-68-SYNT, AUT-QE-SYNT etc.

For a description of AUT-SYNT we refer to appendix 9, a text in AUT-68-SYNT may be found in appendix 8.

Our experiences with translating Landau's book have been a stimulus for developing AUT-SYNT, and have indicated the way this could be done. As no verifying program for SYNT languages was available until after the translation was finished, the SYNT-facility could not be used in the translation. This may be considered unfortunate, because the presence of this facility would have simplified both the writing and the reading of our text.

4.1.1. η-reduction in AUTOMATH

In AUT-68 and AUT-QE one of the possible ways to establish definitional equality is by η-*reduction* ([vD, 6.2.2]): If x is not free in A then $[x,\alpha]\langle x\rangle A >_\eta A$. As can be seen in the list in 3, η reduction was applied only twice during the verification of our translation. We give the lines which required these η-reductions, together with their relevant contexts.

The following lines from the text on propositional logic are presupposed:

```
    * con  :=        PN        E prop
    * a    :=        —         E prop
a   * not  :=     [x,a]con     E prop
a   * u    :=        —         E not(not(a))
u   * et   :=        PN        E a
a   * u    :=        —         E con
u   * cone := et(a,[x,not(a)]u) E a
```

The first line where η-reduction is required occurs in the text on predicate logic. In this text the following lines appear:

* S	:=	—	E type
S * P	:=	—	E [x,S]prop
P * all	:=	P	E prop
P * non	:=	[x,S]not(<x>P)	E [x,S]prop
P * u	:=	—	E not(all(S,P))
u * v	:=	—	E non(non(P))
v * s	:=	—	E S
s * t1	:=	et(<s>P,<s>v)	E <s>P
v * t2	:=	<[x,S]t1(x)>u	E con

In order to verify that the middle part of this last line is a correct expression, it should be established that

$$CAT([x,S]t1(x)) \stackrel{D}{=} DOM(u)$$ (Cf. [vD, 6.2.4.6]) .

We have

$$CAT([x,S]t1(x)) = [x,S]CAT(t1(x)) = [x,S][s/x]<s>P = [x,S]<x>P ,$$

$$DOM(u) = DOM(not(all(S,P))) = DOM([x,all(S,P)]con) = all(S,P) = P .$$

The question is to establish

$$[x,S]<x>P \stackrel{D}{=} P .$$

This obviously requires η-reduction.

The second case in which η-reduction is used occurs in the text on generalized implication:

* a	:=	—	E prop
a * b	:=	—	E [x,a]prop
b * imp	:=	b	E prop
b * u	:=	—	E not(a)
u * th2	:=	[x,a]cone(<x>b,<x>u)	E imp(a,b)

Here, in order to verify the last line, it is asked whether the category of the middle part definitionally equals the category part, i.e. whether

$$CAT([x,a]cone(<x>b,<x>y) \stackrel{D}{=} imp(a,b) .$$

Now

$$CAT([x,a]cone(<x>b,<x>u)) = [x,a]CAT(cone(<x>b,<x>u)) = [x,a]<x>b ,$$

$$imp(a,b) = b$$

and therefore η-reduction must be used for establishing

$$[x,a]\langle x\rangle b \overset{D}{=} b \ .$$

It has been observed by v. Daalen that η-reduction might have been avoided in both cases by a slight modification of the definitions: for all (in the first case) and for imp (in the second case). In fact all might have been defined by

$$P * all := [x,S]\langle x\rangle P \ \underline{E} \ \underline{prop}$$

and imp by

$$b * imp := [x,s]\langle x\rangle b \ \underline{E} \ \underline{prop}$$

This would have made no difference to the rest of the book, apart from the fact that in some places an extra β-reduction would have been necessary. In fact, if a predicate P is defined explicitly (as opposed to being a predicate variable or a primitive predicate constant) then $P = [y,S]m(y)$, say, and we have, without η-reduction

$$[x,S]\langle x\rangle P = [x,S]\langle x\rangle[y,S]m(y) = [x,S][[y/x]]m(y) = [x,S]m(x) = P \ .$$

We conclude therefore that η-reduction does not add considerably to the expressive power of AUTOMATH.

4.1.2. prop v. type

In the stage of exploration of the possibilities to represent logic in AUT-QE, initially a variant of this language was used which did not contain the 1-expression prop . It was therefore impossible to prescribe whether types had to be interpreted as assertion types (containing proofs) or "ordinary" types (containing "ordinary" objects).

Contradiction was represented as a primitive type, negation and the double negation law were formalized in terms of this type as follows:

* con	:=	PN	E type
* a	:=	—	E type
a * not	:=	[x,a]con	E type
a * u	:=	—	E not(not(a))
u * d.n.l.	:=	PN	E a

If in this text a is interpreted as an "ordinary" type, nat say, then expressions of type not(a) (or [x,a]con) could be interpreted as proofs that a is empty (in fact, if we have p E not(a) , then for an object x E a we have ⟨x⟩p to prove contradiction). Hence expressions of type not(not(a))

have to be interpreted as proofs that a is (in a weak sense) nonempty. Given such a proof q we have an object d.n.l(a,q) E a . Or, in other words: d.n.l acts as a Hilbert operator, selecting an object from any nonempty type. In particular this induces a form of the axiom of choice.

As we did not want the double negation law to have such far-reaching consequences, we extended the language by admitting prop as a basic 1-expression. Thus we obtained the language AUT-QE (as defined in [vD, 5]), in which it is possible to distinguish between assertion types and ordinary types.

The distinction of prop and type not only unlinked the double negation law from the axiom of choice, but also made it possible to implement irrelevance of proofs (cf. 4.0.1, 1.3). This opportunity was not seized in the logic underlying our translation (though this would have been natural). For an explanation we refer to 4.2.0.

We may conclude that the distinction between proofs and "ordinary" objects is an essential feature when representing classical logic in AUTOMATH. For representing constructive logic the version with only type keeps its value.

4.1.3. Strings and telescopes

In chapter 2 of his book Landau uses pairs (x_1,x_2) of natural numbers. He considers such a pair as a single object and yet he describes it by two variables. A faithful translation of this practice could have been given if the concept of a *string of expressions* would have been present in our language.

Another use strings of expressions might have is as arguments of partial functions (as described in 1.3). In fact such functions are applied to pairs (a,p) where a is an object of a certain type S , and p a proof that a satisfies some predicate P on S (which describes the range of the function).

As a further example we consider the concept of a group, which might be considered as a string (S,op,iv,e,p) where S E type, op E [x,S][y,S]S, iv E [x,S]S, e E S and p E groupaxioms(S,op,iv,e) .

We usually want the types of the expressions of such a string to satisfy certain conditions. In the case of the argument (a,p) of partial function we want a E S, p E <a>P . In other words we want the argument (a,p) to be consistent with the "abstractor part" of the function: x E S ; y E <x>P . In the case of the group we want a group (S,op,iv,e,p) to be

consistent with

x E type; y E [s,x][t,x]x; z E [s,x]x; u E x;

v E groupaxioms(x,y,z,u)

There is a strong analogy with the case where expressions $A_1,\ldots,A_n$ are required to be suitable candidates for substitution for the variables $x_1,..,x_n$ of a certain context $x_1 \underline{E} \alpha_1, x_2 \underline{E} \alpha_2,\ldots,x_n \underline{E} \alpha_n$ (Cf. [vD, 2.5]).

To describe such conditions on strings we introduce the following terminology. A finite sequence of $\underline{E}$ formulas $x_1 \underline{E} \alpha_1,\ldots,x_n \underline{E} \alpha_n$ is called a *telescope*. The string of expressions $(a_1,\ldots,a_n)$ is said to *fit* into the telescope $x_1 \underline{E} \alpha_1,\ldots,x_n \underline{E} \alpha_n$ if $a_1 \underline{E} \alpha_1, a_2 \underline{E} [\![x_1/a_1]\!]\alpha_2,\ldots,$ $a_n \underline{E} [\![x_1,\ldots,x_{n-1}/a_1,\ldots,a_{n-1}]\!]\alpha_n$.

Extension of the language with constants and variables for strings and defined constants for telescopes has been proposed by de Bruijn. This is especially helpful when formalizing abstract structures such as groups, vector spaces or categories, and has been applied on a large scale by J. Zucker (Cf. [Z]).

4.2. Comments on the translation

In this section we first give a chronological survey of the different representations of logic which have been tried, and we state the motives for finally choosing AUT-QE as a language for our translation. Furthermore we mention some aspects which are (in our opinion) shortcomings of the translation and we add some positive conclusions which can be drawn from our work.

4.2.0. Choice of the language

In our first attempts to translate Landau's "Grundlagen" in AUTOMATH, we used the language AUT-68. The representation of logic was similar to the one described in 4.0.2 and presented in appendix 6. Elimination and introduction of $\forall$ were effected by the axioms $\forall$e (with parameters S E type, p E [x,S]PROP, a E S, u E ⊢(∀(S,P))) and $\forall$i (with parameters S E type, P E [x,S]PROP, u E [x,S]⊢(<x>P)). These axioms were used frequently in developing logic, because the logical connectives and the existential quantifier were defined in terms of $\forall$. On the basis of this logic chapter 1 of Landau's book was translated in AUT-68.

At that stage of our work we started trying to represent logic in AUT-QE, initially using a variant of that language which did not contain prop. In AUT-QE the axioms ∀i and ∀e were superfluous: if P E [x,S]type (i.e. P represents a predicate on S) then objects of type P can be interpreted as proofs of ∀(S,P) . Conversely, given such an object u E P and an object a E S we have <a>u E <a>P (i.e. <a>u proves that P holds at a). As a consequence the text on logic in AUT-QE was considerably shorter then the earlier text in AUT-68. (It was not observed at that time, that this was caused essentially by the redundant parameters S and P of both constants ∀e and ∀i .) So AUT-QE seemed to be a much better language, and therefore a fresh start was made with the translation of Landau's book into that language. In 4.1.2 we have reported that in this system (AUT-QE without prop) the double negation law induces a Hilbert operator. This led us to add prop as a basic 1-expression to our language, thus extending it to proper AUT-QE.

At the time we finally fixed the language we did not appreciate the fundamental importance of incorporating a form of irrelevance of proofs. This was due mainly to two reasons:

i) Partial functions are not frequently used in the first three chapters of Landau's book, and for those partial functions which are defined there, irrelevance of proofs could be derived. Therefore no need was felt for an axiom.

ii) As Landau, being a classical mathematician, does not discuss proofs at all, we thought we should try to follow this practice. Consequently we did not want to have an axiom declaring proofs equal.

4.2.1. Shortcomings of the translation

Here I list those features of the translation which I would change if I were to redo the work.

i) In my opinion the SYNT-facility should be present in any AUTOMATH language. It will bring texts in AUTOMATH closer to mathematical practice. The middle parts of many lines in the present Landau translation are unnecessarily complex and tedious (both to the reader and to the writer). because this facility is absent in the language I used.

ii) I regret that I have not implemented irrelevance of proofs as an axiom. As I see it now, for representing classical reasoning a language should be chosen which even contains irrelevance of proofs by definitional

equality (Cf. 4.0.2).

iii) Some of the names I have used lack expressive power. This is partly due to the fact that AUT-QE admits only alphanumeric identifiers, but mainly to my excessive preference for short names.

iv) I am not content with the translation of chapter 5, § 8. This text is overloaded with irrelevant embedding and lifting functions which hamper a clear understanding of the argument. I think it is better to define $\sum_{i=1}^{n} f(i)$ and $\prod_{i=1}^{n} f(i)$ for functions f defined for all natural numbers (and not just on an initial part of the naturals), although this procedure deviates slightly from Landau's intentions.

4.2.2. Final remarks

The main positive comment we can make on the translation is that it has been succesfully finished (in spite of some inconveniences in the language).

An aspect which has not been mentioned so far is the ratio between the length of pieces of AUT-QE text and the length of the corresponding German texts. Our claim at the outset was that this ratio can be kept constant. We give a few data. As pieces of text we have chosen the chapters of Landau's book, and as a measure of the lengths the number of stored AUT-QE expressions (storing expressions requires storing all subexpressions too) and (rough estimates of) the number of German words (where "x" and "+" were counted as words). We give the following list:

	chapter 1	chapter 2	chapter 3	chapter 4	chapter 5
no. of expressions	12200	25800	30300	35000	60500
no. of words	3200	4900	5300	5500	11000
$\frac{\text{no. of expressions}}{\text{no. of words}}$	3,8	5,3	5,7	6,4	5,5

The high ratio in chapter 4 might be attributed to the complicated definitions by cases in this chapter, while the low ratio in chapter 1 is possibly caused by the absence of calculations.

Another notable aspect of the work is the comparatively small place taken by the preliminaries. It appears that a formal treatment of the logic underlying mathematics (if we disregard metalogic) is much easier than a formal treatment of mathematics itself.

It has not been the purpose of this enterprise to construct a formal system which suits my own fancy and to develop in this system the theory of naturals, reals and complex numbers. I have rather tried to represent in a language which was essentially given beforehand, a wide variety of concepts and ideas as expressed in a book like Landau's. The success of this undertaking is due to the flexibility of AUTOMATH languages, and to the close connection which can be made between these languages and intuitive human reasoning.

Appendix 1.

REPRINT. Published in the
Proceedings of the Symposium
on APL (Paris, December 1973),
ed. P. Braffort.

A description of AUTOMATH and some aspects of its language theory

by

D.T. van Daalen *)

0. Summary

This note presents a self-contained introduction into AUTOMATH, a formal definition and an overview of the language theory. Thus it can serve as an introduction to the papers of L.S. Jutting [7] and I. Zandleven [11] in this volume**). Among the various AUTOMATH languages this paper concentrates on the original version AUT-68 (because of its relative simplicity) and one extension AUT-QE (in which most texts have been written thus far).
The contents are:

1. Introductory remarks.
2. Informal description of AUT-68.
3. Mathematics in AUTOMATH: propositions and types.
4. Extension of AUT-68 to AUT-QE.
5. A formal definition of AUT-QE.
6. Some remarks on language theory.

For a description of the AUTOMATH project and for its motivation we refer to Prof. De Bruijn's paper also in this volume [4].

*) The author is employed in the AUTOMATH project and is supported by the Netherlands Organization for the Advancement of Pure Science (Z.W.O.).

**) In the present appendix "this volume" refers to: Proceedings of the Symposium on APL (Paris, December 1973) ed. P. Braffort.

1. Introductory remarks

1.1. According to the claims for the *formal system* AUTOMATH one should be able to formalize many mathematical fields in it in such a precise and complete fashion that machine verification becomes possible. The flexibility required to meet the indicated universality is provided by having a rather meagre *basic system*. The AUTOMATH user himself has to add appropriate *primitive notions* to the basic system in order to introduce the concepts and axioms specific to the part of mathematics he likes to consider. In this respect, the basic system may be compared with some usual system of logic (e.g. first order predicate calculus) to which one adds mathematical axioms in order to form mathematical theories.

1.2. In spite of this analogy however the basic system itself does not contain any logic in the usual sense. Basic for the system are the concept of *type* and *function* (instead of, e.g., the concept of set or of natural number), which are formalized by a certain *typed λ-calculus*.

When representing mathematics in AUTOMATH one has to deal with the question of *coding:* How to formalize general mathematical concepts in the form of *types* and *functions* (see section 2.2). Clearly an appropriate formalization will incorporate as much as possible of the basic type-and-function framework. Section 3 discusses this coding problem and in particular proposes a suitable way of representing propositions, predicates and proofs (a *functional interpretation* of logic).

1.3. In order to satisfy the claim of automatic verification of correctness the system certainly has to be decidable (and even *feasibly decidable* on now-existing computing machines). Since many common mathematical theories produce undecidable sets of theorems we must conclude that we cannot expect the computer to do all our work. Indeed theorems have to be given *together with their proofs* in order to allow verification.

Thus the correctness produced by the machine verification covers the arguments leading from axioms to conclusions only. The AUTOMATH user himself is responsible for his choice of primitive notions and all the coding (and decoding) involved.

2. Informal description of AUTOMATH

2.1. Introduction

Here we treat the original version of AUTOMATH, now named AUT-68. We chose this system as an example because of its relative simplicity. The discussion will be informal and intuitive and in fact restricted to the object-and-type fragment of the language (thus leaving the proof-and-proposition fragment to section 3).

2.2. Intuitive framework

(This section may be skipped by formalists).

The mathematical entities discussed in the language fall into two sorts: *objects* and *types*. The types may be considered as classes or sets of a certain kind, which may have objects as their elements. All types are supposed to be disjoint, for each object belongs to just one type. This *uniqueness of types* permits one to speak about *the* type of an object.

The typestructure is built up by starting from *ground types* and forming *function types* from these. Each mathematician may choose the ground types himself (as primitive notions), e.g. the type of natural numbers.

An example of a function type is the type $\alpha \to \beta$ (where α and β are types) of the functions from α to β. More generally, the function types are formed by taking *products*, as follows: The language allows one to express dependence of types on objects (of some given type). That is, one can describe certain families of types β_x indexed by the objects x of a given type α. Now every function type is formed as the *generalized Cartesian product* of such β_x, usually denoted $\prod_{x \in \alpha} . \beta_x$, and containing as objects just these functions that associate to any object x of type α an object of type β_x. The type $\alpha \to \beta$ is the special case where all β_x are a fixed type β.

2.3. Expressions, degrees and formulas; correctness

The language as such only expresses the constructions of types and objects and the typing relations between objects and types.

The *expressions* of the language have *degree* 1, 2 or 3. Types and objects are denoted by expressions of degree 2 and 3 respectively (for short 2-expressions, 3-expressions). For convenience we introduce the 1-expressions type to provide a type for the types. Further 1-expressions will be introduced in sections 3 and 4.

The symbol $\underline{E}$ expresses the typing relation: ... has type ... . So if A denotes an object then we have the E-*formulas* A $\underline{E}$ α and α $\underline{E}$ type. The 2-expressions and 3-expressions are built up from *variables* and *constant-expressions* by means of:

i) the substitution mechanism (section 2.5)

ii) functional abstraction and application (sections 2.8 and 2.10).

The constant-expressions have the form $c(x_1,\ldots,x_k)$ where $x_1,\ldots,x_k$ are variables and c is either a *primitive* constant introduced as a primitive notion (section 2.6) or a *defined* constant (section 2.7).

Expressions and formulas are *correct* if they are constructed according to the rules of the language, which are informally discussed in the sequel.

2.4. Variables and contexts

A mathematical statement generally presupposes certain assumptions on the variables used. For example: "let x be a natural and y a real number". In AUTOMATH, in accordance whis this usage, each variable of degree 3 *(object-variable)* ranges over a certain type, called the type of the variable. The 2-variables *(type-variables)* are supposed to range through the types and have type as their type.

Expressions and formulas containing *free* object- or type-variables, say $x_1,\ldots,x_k$, can only be *correct* relative to a certain *context*: I.e. a finite sequence of $\underline{E}$-formulas x_1 $\underline{E}$ $\alpha_1,\ldots,x_k$ $\underline{E}$ α_k, called *assumptions*, in which the free variables have to be explicitly introduced with their types.

Some of the types α_i may depend on the variables given earlier in the sequence. For instance, α_3 may contain both x_1 and x_2 as free variables. It is understood that all α_i are correct expressions themselves: α_1 relative to the *empty* context, α_2 relative to x_1 $\underline{E}$ α_1, etc.

2.5. Substitution mechanism

Let us, in informal discussion, exhibit the possible dependence of an expression Σ on variables $x_1,\ldots,x_k$ by writing $\Sigma[\![x_1,\ldots,x_k]\!]$ for Σ. Then we write $\Sigma[\![A_1,\ldots,A_k]\!]$ for the result of *simultaneously substituting* A_i for x_i (for $i = 1,\ldots,k$) in Σ.

Suppose that under assumptions x_1 $\underline{E}$ $\alpha_1,\ldots,x_k$ $\underline{E}$ α_k we have a correct $\underline{E}$-formula $A[\![x_1,\ldots,x_k]\!]$ $\underline{E}$ $\alpha[\![x_1,\ldots,x_k]\!]$. Then the *substitution mechanism* yields the substitution *instance* $A[\![A_1,\ldots,A_k]\!]$ $\underline{E}$ $\alpha[\![A_1,\ldots,A_k]\!]$ for any sequence

$A_1,\ldots,A_k$ of suitable candidates for $x_1,\ldots,x_k$. I.e. these $A_1,\ldots,A_k$ have to be of the appropriate types where, however, in view of the possible dependence of types on variables, the substitution has to take place in the types too. So we require

$$A_1 \underline{E} \alpha_1,\ A_2 \underline{E} \alpha_2[\![A_1]\!],\ldots,A_k \underline{E} \alpha_k[\![A_1,\ldots,A_{k-1}]\!] .$$

2.6. Primitive notions

As mentioned before, one has to add primitive notions to the basic system in order to introduce the specific concepts of the piece of mathematics one wants to study.

For example, in order to write about the natural numbers, one might introduce the primitive *type-constant* nat and the *object-constant* 1 by axiomatically stating:

$$\text{nat } \underline{E}\ \underline{\text{type}}$$

$$1\ \underline{E}\ \text{nat} .$$

In general, primitive notions are introduced by stating an axiomatic $\underline{E}$-formula $p(x_1,\ldots,x_k)\ \underline{E}\ \alpha[\![x_1,\ldots,x_k]\!]$ under certain assumptions $x_1\ \underline{E}\ \alpha_1,\ldots x_k\ \underline{E}\ \alpha_k$. Here either α is $\underline{\text{type}}$ (and p is a type-constant) or in the current context we have $\alpha\ \underline{E}\ \underline{\text{type}}$ already (p being an object-constant).

All correct substitution instances $p(A_1,\ldots,A_k)$ of such a constant-expression $p(x_1,\ldots,x_k)$ can be produced by the substitution mechanism, described above.

For example, the concept of *successor* in the natural number system can be introduced under the assumption $x\ \underline{E}$ nat by stating: successor(x) $\underline{E}$ nat.

Using the substitution mechanism we get

$$\text{successor}(1)\ \underline{E}\ \text{nat}$$
$$\text{successor}(\text{successor}(1))\ \underline{E}\ \text{nat, etc.}$$

Notice that primitive constant-expressions may not only contain object-variables (like the x in successor(x)) but also type-variables.

2.7. Abbreviations

In mathematics one often introduces abbreviations, i.e. new names for possibly long and complicated expressions. In AUTOMATH this abbreviation facility is also present; indeed, it will appear that by the particular format of the language every *derived* statement gives rise to the introduction

of a new defined constant. Although this kind of explicit definition is often considered theoretically uninteresting, we feel that it is essential in practice for the actual formalization and verification of complicated theories.

Just like primitive notions, abbreviations are introduced under certain assumptions and so may contain free variables in general. Thus new constant-expressions $d(x_1,\ldots,x_k)$ are introduced, abbreviating expressions D which are correct in the current context. Clearly the type of $d(x_1,\ldots,x_k)$ must be the same as that of D.

Example: 2,3,... can be introduced by

2 := successor(1)

3 := successor(2), etc.

Further, the notion of "successor of successor" might be abbreviated by stating (under assumption x E nat) that

plustwo(x) := successor(successor(x)) .

Again, all correct substitution instances with their types can be produced by the substitution mechanism.

2.8. Functional abstraction: λ-calculus

We have mentioned functional abstraction and application as further tools for constructing expressions. By these devices a form of typed λ-calculus is incorporated into the basic system. In λ-calculus, intuitively speaking, λx.B denotes the function which to any object x associates the object B. Or (exhibiting the dependence on x) λx.B[x] is the map which, with any A, associates B[A].

In AUTOMATH (where all functions have a *domain)* such explicitly given functions are denoted by *abstraction expressions* [x,α]B, where B may contain x as a free variable; α is the type of x and the domain of the function. In case B is a 3-expression, [x,α]B attaches objects to the objects of type α and is called an *object-valued function*. If B is a 2-expression, [x,α]B attaches types to the objects of type α and is called a *type-valued function*. In AUT-68 no abstraction expressions of degree 1 are formed (in contrast with AUT-QE).

Notice that possible free *occurences* of x in B are *bound* by the abstractor [x,α] and are not free in [x,α]B any more. An important restriction on abstracting is that such a bound variable must be a 3-variable. Thus we only *quantify* (cf. section 3.4) over (the objects of) a given type and quantifica-

tion over type is not possible.

2.9. Type of abstraction expressions

Suppose that under the assumption x E α we have B E β. If β is not a 1-expression then we may form both the abstraction expressions [x,α]B and [x,α]β. According to section 2.8 [x,α]B denotes an object-valued function and [x,α]β denotes a type-valued function.

The latter abstraction expression [x,α]β[x] however is also used with a different meaning in AUTOMATH, that is, to denote the *corresponding function type* $\prod_{x E \alpha} \cdot \beta[x]$ (which is the type of [x,α]B[x] by section 2.2).

So we obtain [x,α]B E [x,α]β and [x,α]β E type.

Example: the successor *function* can be introduced (in the empty context) by

succfun := [x,nat]successor(x) E [x,nat]nat .

The double use of 2-expressions mentioned above does not cause ambiguity, because it is always clear whether an expression acts as a function or as a type in a formula. In fact in AUT-68 abstraction expressions of degree 2 are exclusively used with the second meaning, i.e. as function types.

2.10. Functional application

In full (i.e. type-free) λ-calculus any expression - as a function - may be applied to any expression - even itself - as an argument.

In AUTOMATH, as a *typed* λ-calculus, all functions have *domains* and any form of self-application is ruled out by the *application restrictions:* The *application expression* <A>B (denoting the result of applying B as a function to A as an argument) is correct only if:

i) B is a function and so has a domain, say α.

ii) A is an object of type α.

The notation <A>B, with the argument in front, is somewhat unusual; it is convenient however since abstractions are written in front too.

2.11. Type of application expressions

Assume that B E [x,α]β. Here [x,α]β[x] is a 2-expression acting as a type and so denotes $\prod_{x E \alpha} .\beta[x]$. Hence B must be considered as a function with domain α. Now if A E α we are allowed to form the application expression <A>B having β[A] as its type.

Note that B need not be of the form [x,α]C itself. It may, e.g., be a single object variable or object constant with type [x,α]β.
Example: As an alternative expression for the number 3 we might introduce

3alt := <2>succfun E nat .

2.12. Equality

We will define a relation of *definitional equality* among the correct expressions, appropriate to the interpretation of expressions suggested above. The relation is denoted ... = ... and generated by:

i) *abbreviational* or δ-*equality*, $=_\delta$
ii) λ-*equality*.

The latter is generated in turn by β-*equality*, $=_\beta$, and η-equality $=_\eta$. Usually in λ-calculus the λ-equality also explicitly embodies α-*equality* (renaming of bound variables). In this note however we take the point of view of simply ignoring the names of the bound variables. So α-equal expressions are identified and are a fortiori definitionally equal by the reflexivity of the = -relation (cf. also section 5.3.2).

2.12.1. δ-equality

Assume the defined constant d has been introduced in suitable context by

$$d(x_1,\ldots,x_k) := D[\![x_1,\ldots,x_k]\!] .$$

Then $d(x_1,\ldots,x_k)$ abbreviates D and we write $d(x_1,\ldots,x_k) =_\delta D$. And further for the substitution instances:

$$d(A_1,\ldots,A_k) =_\delta D[\![A_1,\ldots,A_k]\!] .$$

2.12.2. β-equality

Assume <A>[x,α]B[x] is a correct expression (so A E α). Now β-equality exploits the interpretation of [x,α]B as a function with domain α and simply amounts to evaluating the result of the application:

$$<A>[x,\alpha]B =_\beta B[\![A]\!] .$$

2.12.3. η-equality

In mathematics one usually considers functions as *extensional* objects,

in the sense that functions with the same domain and which are pointwise equal are identified. In AUTOMATH this extensional equality is *partly* covered by the η-equality: *If* x *does not occur free in* B *then* $[x,\alpha]\langle x\rangle B =_\eta B$ (for correct expressions only). This is intuitively sound only if domain B = α, which indeed is the case by the correctness of $[x,\alpha]\langle x\rangle B$.

2.12.4. Definitional equality

Now definitional equality = is defined to be the *equivalence relation* on the correct expressions, generated by $=_\delta$, $=_\beta$, $=_\eta$ and by *monotonicity: If* A = A' *and* B' *is produced from* B *by replacing one specific occurrence of* A *in* B *by (an occurrence of)* A' *then* B = B'.

Or, using *suggestive dots* for the *unchanged* part of the expression B: *If* A = A' *then* ...A... = ...A'... .
Example of the monotonicity rule: If A = A' then <C><A>D = <C><A'>D (if both expressions are correct).

2.13. The format: books and lines

2.13.1. Actual AUTOMATH texts are written in the form of books. A *book* consists of a finite sequence of *lines*. Each line must be placed in a certain *context* (the context of the line) and introduces a new identifier of a certain type. All lines consist of four consecutive parts, separated by suitable marks or spaces:

i) *context part,* indicating the context of the line. In general the context part consists of the *context indicator,* i.e. the last variable of the current context. From this the complete context can easily be recovered. If the context of the line is $x_1 \underline{E} \alpha_1,\ldots,x_k \underline{E} \alpha_k$, the sequence of variables $x_1,\ldots,x_k$ is called the *indicator string* of the line. The *empty* context can be indicated by an empty context part.
ii) *identifier part,* consisting of the new *identifier.*
iii) *middle part,* containing the symbol EB (cf. 2.13.2), the symbol PN (cf. 2.13.3) or the *definition* of the new identifier (cf. 2.13.4).
iv) *category part,* containing the type of the new identifier.

Assume an AUTOMATH book is given, in which the variable x_k has been introduced with type α_k in the context $x_1 \underline{E} \alpha_1,\ldots,x_{k-1} \underline{E} \alpha_{k-1}$. Then we may add lines with context indicator x_k, so having $x_1 \underline{E} \alpha_1,\ldots,x_k \underline{E} \alpha_k$ as their context. Below we discuss the three different kinds of lines.

2.13.2. The *block opening lines* have middle part EB (for *empty block opener)* or, in alternative notation, a bar ——— . An EB-line introduces a new *variable* and thus allows extension of the current context by one assumption. Example: x_k * y := EB E α ("let y be of type α") introduces a new variable y of type α. Lines having y as their context part - which may appear later in the book - then have x_1 E α_1,...,x_k E α_k, y E α as their context.

2.13.3. The *primitive notion lines* have middle part PN and introduce the primitive notions. For example:

$$x_k * p := PN\ E\ \alpha$$

introduces the primitive constant expression $p(x_1,\ldots,x_k)$ and contains the *axiomatic* E-statement $p(x_1,\ldots,x_k)$ E α.

2.13.4. The *abbreviation lines* look like:

$$x_k * d := D\ E\ \alpha\ ,$$

where the middle part D is the definition of d, i.e. the expression to be abbreviated. This line contains, relative to the preceding book and the current context, both the derived E-statement D E α and the defining axiom for the new defined constant d:

$$d(x_1,\ldots,x_k) := D\ .$$

2.14. Correctness of lines; validity

A line is *correct* if both the middle part (if not EB or PN) and the category part are correct expressions with respect to the preceding book and the current context, and the category part is the type of the middle part (if not EB or PN). For the correctness of the expressions, all identifiers used have to be *valid*. Constants are valid in a book from the line on in which they are introduced. Free variables are valid in a line if they occur in its context. We speak about the *block* of lines in which a free variable is valid (whence *block opener)*.

2.15. Shorthand facility

Assume that a primitive or defined constant c was introduced in a certain context x_1 E α_1,...,x_k E α_k. Then if later in the book c occurs with fewer than k arguments, the argument list is completed by adding a suitable

initial segment of the original indicator string (cf. 2.13.1ii)) $x_1,\ldots,x_k$. In other words the expression $c(A_{i+1},\ldots,A_k)$ is shorthand for $c(x_1,\ldots,x_i,A_{i+1},\ldots,A_k)$ and the single constant c is shorthand for $c(x_1,\ldots,x_k)$. Clearly the completing variables have to be valid, that is, the initial segments of the original and the current context have to coincide. The shorthand facility accords with usual mathematical practice where free variables are often considered as fixed throughout an argument and are not mentioned explicitly.

2.16. Paragraph system

For each variable and constant it must be possible to retrace from which line it originates. This condition is clearly satisfied when all names are unique. A more liberal method of naming however is allowed by the socalled *paragraph system*, for a description of which we refer to Zandleven [11, section 11]. Both shorthand facility and paragraph system do not really concern the language definition but are present for convenience only.

2.17. Example

In the following AUT-68 booklet the examples of the preceding sections are now written in the proper format.

```
    * nat        := PN                    type
    * 1          := PN                    nat
    * x          := ———                   nat
  x * successor  := PN                    nat
    * 2          := successor(1)          nat
    * 3          := successor(2)          nat
  x * plustwo    := successor(successor)  nat
    * succfun    := [x,nat]successor(x)   [x,nat]nat
    * 3alt       := <2>succfun            nat
```

Here the middle part of plustwo uses the shorthand facility. It is left to the reader to establish 3 = 3alt.

3. Mathematics in AUTOMATH: Propositions as types

3.1. Functional interpretation of logic

Up till now we have described AUTOMATH as a calculus of objects and their types only. A major part of mathematics however consists of making statements and reasoning with them, i.e. deals with logic.

Now there are different ways of coding some logic into the objects-and-types framework. Here we only mention a socalled *functional interpretation* of logic, which gives rise to the *propositions-as-types* notion. This idea of interpreting logic was developed independently by de Bruijn and certain others, of whom we mention Howard [6], Prawitz [10], Girard [5] and Martin-Löf [8].

3.2. Propositions as types

So far we have introduced type as the only 1-expression. We had Σ E type and Γ E Σ for the types Σ and the objects Γ of type Σ respectively. Now we introduce another 1-expression, the basic symbol prop. Originally in AUT-68 no distinction was made between type and prop. The latter 1-expression acts just like type and was introduced later to allow difference of treatment between types which are to be considered as propositions and types which are just types of objects.

If Σ E prop we consider Σ as a proposition. If further Γ E Σ , we consider Γ as some construction establishing the truth of Σ (a "proof" of Σ). Thus the formula Γ E Σ is conceived as *asserting* the proposition Σ.

3.3. Interpreting implication

Let α E prop and β E prop. Now we may say we have a "proof" of the implication $\alpha \to \beta$ if from an assumption of the truth of α we can argue and conclude the truth of β. That is, if for any construction establishing the truth of α we can produce a construction for the truth of β or, equivalently, if we have a map from "proofs" of α to "proofs" of β.

Now in AUTOMATH terminology: we say we "prove" $\alpha \to \beta$ if for any x E α we can produce some B E β. I.e. if we have some Σ in the function type $[x,\alpha]\beta$. So we let $[x,\alpha]\beta$ denote the implication $\alpha \to \beta$ and have $[x,\alpha]\beta$ E prop. This corresponds to the second interpretation of abstraction expressions in section 2.9.

Now by this interpretation we obtain the *modus ponens* (from α and $\alpha \to \beta$ infer β) by simple functional *application*. For let A E α and Σ E[x,α]β (A and Σ thus being "proofs" of α and $\alpha \to \beta$ respectively). Then by the application rule we construct <A>Σ establishing the truth of β.

3.4. Universal quantification; negation

In exactly the same manner a function interpretation of *universal* statements can be given. Namely if α E type and for x E α we have β E prop then we identify the function type [x,α]β with the universal statement $\forall_{x \in \alpha}\beta$. Here functional application corresponds to the *"instantiation" rule* in logic.

Thus by this interpretation of logic in AUTOMATH one gets the $(\forall,\to)$-fragment of first order predicate logic for free. However in AUTOMATH only positive statements are made and statements like: "Σ is not of type Γ" cannot be expressed. In order to interpret negation we introduce as a primitive notion the proposition *con* (for "contradiction") together with some suitable axiom (primitive notion). Here are different possibilities, e.g. the intuitionistic *absurdity rule* (for any proposition α, from *con* infer α) or the classical *double negation law*. Then an AUTOMATH theory (i.e. book) is *consistent* if, in the empty context, it does not produce some Σ E *con*.

For α E prop we define *non*(α) as $\alpha \to$ *con* or, in AUTOMATH notation, [x,α]*con*. Now the double negation law can be stated by introducing the primitive notion *dnl* as follows: If α E prop, x E *non(non(α))* then *dnl*(α,x) E α.

By also choosing suitable definitions for the other connectives $(\wedge,\vee)$ and the existential quantifier we can smoothly obtain full classical first order predicate calculus.

3.5. Assumptions, axioms, theorems

In AUTOMATH-books the E-formula Γ E Σ for a proposition Σ can occur in the usual three kinds of lines again:

i) EB-lines: σ * x := EB E Σ.
   These must be interpreted as *assumptions:* "let Σ hold" or "let x be a proof of Σ". Now in a line where x is valid we may refer to x whenever we want to use the assumed truth of Σ.

ii) PN-lines: σ * p := PN E Σ.
   These serve as axioms, or rather as axiom *schemes* (by the dependence on the variables contained in the context σ).

iii) abbreviation lines: σ * d := Γ E Σ must be considered as derived state-

ments, i.e. theorems, lemmas etc. Here the middle part Γ "proves" the proposition Σ from the assumptions in the context σ.

3.6. Book-equality

The definitional equality (cf. section 2.12) of AUTOMATH only covers a small part of the usual mathematical equality. Further a statement of definitional equality cannot be handled as an actual proposition; e.g. it cannot be negated or even assumed (as in: let A = B). As the AUTOMATH-counterpart of the usual mathematical ...equals... the *book-equality* IS(α,A,B) - where A and B are objects of type α - can be introduced by suitable primitive notions, some of which are shown in the example below.

```
  * α       := ——        type
α * x       := ——        α
x * y       := ——        α
y * IS      := PN        prop
x * REFL    := PN        IS(x,x)
y * i       := ——        IS(x,y)
i * SYM     := PN        IS(y,x)
           etc.
```

and also:

```
α * β       :=  ——       type
β * f       :=  ——       [x,α]β
f * x       :=  ——       α
x * y       :=  ——       α
y * i       :=  ——       IS(x,y)
i * ISAX1   :=  PN       IS(β,<x>f,<y>f)
```

By the axiom of reflexivity (REFL) above, definitional equality implies book-equality: if A E α, B E α, A = B then REFL(α,A) E IS(α,A,B).

4. Extension of AUT-68 to AUT-QE

4.1. Function-like expressions

Expressions Σ such that Σ E $[x,\alpha]\beta$ or $\Sigma = [x,\alpha]\beta$ are called *function-like* expressions. Whereas in AUT-68 function-like 3-expressions may have any form, e.g. they can be variables or primitive constant expressions, the only function-like 2-expressions are (possibly abbreviated) abstraction expressions. This is because function-like 1-expressions are absent in AUT-68.

Thus we can discuss explicitly constructed families of types β_x where x ranges over some type α (namely by forming the abstraction expression $[x,\alpha]\beta[x]$) but we cannot discuss *arbitrary* families of types indexed by x E α. Indeed, we cannot introduce a family of types as a primitive notion or as a variable.

4.2. Supertypes or quasi-expressions

In AUT-QE such arbitrary type-valued functions are admitted however, by extending the class of 1-expressions. The new 1-expressions, *quasi-expressions* (whence AUT-*QE*) or *supertypes*, have the form $[x_1,\alpha_1]$ ... $[x_k,\alpha_k]$ type or $[x_1,\alpha_1]...[x_k,\alpha_k]$ prop, where $\alpha_1,...,\alpha_k$ are 2-expressions, i.e. propositions or types.

For example, an arbitrary type-valued function on α can be introduced by an EB-line:

$$\sigma * f := \text{——} \quad [x,\alpha]\text{type} .$$

If for α we take the type of natural numbers, then f is an arbitrary *sequence of types*.

4.3. The use of AUT-QE

Similarly we have arbitrary *prop-valued functions* in AUT-QE. These are especially useful in our interpretation of logic, for a prop-valued function with domain α is nothing but a *predicate* over α. For example, by an EB-line

$$\sigma * R = \text{——} \quad [x,\text{nat}][y,\text{nat}]\text{prop}$$

an arbitrary binary predicate (rather: relation) on the natural numbers is introduced. The presence of predicate and relation variables in AUT-QE allows us to write *axiom schemes* with such variables, e.g. to introduce a further equality axiom (cf. section 3.6) we can write:

α * P := —— [x,α]prop
P * x := —— α
x * y := —— α
y * i := —— IS(x,y)
i * j := —— <x>P
j * ISAX2 := PN <y>P

We emphasize however that abstraction over such 2-variables (e.g. type-variables, prop-variables, predicate-variables) in AUT-QE is still forbidden, so both AUT-68 and AUT-QE may still be called *first-order systems*.

4.4. Type-inclusion and prop-inclusion

Just as in AUT-68 the function-like 2-expression f (cf. section 4.2) also codes its corresponding function space, i.e. the type of those g with domain α such that for A E α we have <A>g E <A>f. As prop behaves just like type, the predicate P (cf. section 4.3) also denotes the proposition $\forall_{x E \alpha}.P(x)$.

As a consequence, we allow the transition from Σ E [x,α]type to Σ E type. This transition or, in general, from

$$\Sigma\ E\ [x_1,\alpha_1]\ \ldots\ [x_k,\alpha_k][y_1,\beta_1]\ \ldots\ [y_m,\beta_m]\text{type}$$

to

$$\Sigma\ E\ [x_1,\alpha_1]\ \ldots\ [x_k,\alpha_k]\text{type}$$

is called *type-inclusion*. The similar transition with prop instead of type is called *prop-inclusion*. By this *type-inclusion* and *prop-inclusion* AUT-QE contains AUT-68 as a proper *subsystem*. Notice that for 2-expressions uniqueness of types - if A E α, A E β then α = β - is lost.

4.5. Let us finish with a table in which some AUTOMATH notions are listed with their possible meanings in the propositions-as-types interpretation.

AUTOMATH-notions	object-and-type interpretation	proof-and-proposition interpretation
2-expressions	types	propositions
3-expressions	objects	proofs
... E ...	... has type ...	... proves ...
function-like 2-expressions	type-valued functions	predicates
	function types	implications
		universal statements
EB-lines	variable introductions	assumptions
PN-lines	primitive object introductions	axioms
abbreviation lines	definitions or abbreviations	theorems

5. A formal definition of AUT-QE

5.1. The language, to be defined formally now, is the one accepted by the current checker (cf. [11]) except for two points:

i) Paragraph facilities are not present here so all constant names have to be distinct (cf. section 2.16).

ii) There is no shorthand facility (i.e. all expressions are written out in full (cf. section 2.15).

The actual formalism has been chosen in this way in order to keep as close as possible to the preceding informal book-and-line description. A definition along more usual *natural deduction* lines may possibly be more elegant. For technical reasons we preferred to avoid redundancy almost completely in our definition. As a consequence of this, some useful extra rules follow as *derived rules* in the section on language theory.

5.2. Our aim is to define formally what correct AUT-QE books are.
The description consists of:

i) Preliminaries, mainly devoted to the context free part of the language (section 5.4).

ii) *Simultaneous* definition of correctness of books, contexts, lines, expressions, E-formulas and =-formulas (section 5.5).

The =-formulas only serve as a help in our definition; they do not appear in the book. The kernel of ii) is the definition of correctness of expressions and formulas relative to a certain book and context. Here the book serves to determine the set of primitive notions and abbreviations, and the context serves to determine the set of valid free variables.

Most concepts are introduced by *ordinary inductive definitions*. These consist of a finite set of rules of the form: "if ... then ...". Here only such conclusions may be drawn which follow from a finite number of applications of the rules.

5.3. Notational conventions

5.3.1. An extensive use is made of *syntactic variables* throughout the definition. Often certain assumptions on these variables are implicit by their specific choice, e.g. σ and ξ always run over contexts. Syntactic variables may always be indexed or primed.

5.3.2. As for substitution and α-*conversion* (renaming of bound variables) we adopt the following point of view: expressions with bound variables are considered as named versions - named to facilitate reading - of some actually *namefree* skeleton (cf. [3]). Thus we identify α-equal expressions and assume that α-conversion is applied whenever necessary to avoid *clash of variables*. We use ... ≡ ... to denote *syntactic identity* (symbol-for-symbol equality) modulo α-equality. E.g. [x,Σ]...x...x...≡[y,Σ]...y...y... .

5.3.3. Correctness of expressions A and formulas φ relative to a book $\mathcal{B}$ and a context σ are abbreviated by $\mathcal{B}$; σ ⊢ A and $\mathcal{B}$; σ ⊢ φ respectively. Sometimes we write ⊢ A or σ ⊢ A for $\mathcal{B}$; σ ⊢ A and ⊢ φ or σ ⊢ φ for $\mathcal{B}$; σ ⊢ φ when there is no particular need to emphasize the current book or context. The notations $\vdash^{(i)}$A and $\vdash^{(i)}$A E B are used to express that A is an i-expression and ⊢ A (respectively ⊢ A E B).

5.4. Preliminaries

5.4.1. Alphabet

1) As *variables* and *constants* we allow any *alphanumeric string*. Such a string is considered atomic and is thus counted as one single symbol. Syntactic variables for variables are x,y,z,... . Among the constants (syntactic variable c) we distinguish *primitive* (syntactic variables p,q) and *defined* or *abbreviational* constants (syntactic variable d).
2) Improper symbols
   i) Some *brackets* and *braces*: [,], (,), < , >.
   ii) Some *separation marks*: !, *, ⊢, E, :=, =, *semicolon* and *comma*.
   iii) Some *reserved symbols*: EB, PN.

5.4.2. Expressions (syntactic variables A,B,C,D,...,Σ,Δ,Γ,...)

i) *Variables*: x
ii) *Abstraction expressions*: [x,Σ]Δ
iii) *Application expressions*: <Σ>Δ
iv) *Constant-expression instances*: $c(\Sigma_1,\ldots,\Sigma_k)$
v) *Basic constants*: type, prop.

As special syntactic variables for 2-*expressions* we take α,β,... .

5.4.3. Formulas (syntactic variable φ)

i) $\underline{E}$-*formulas*: $\Sigma \mathrel{\underline{E}} \Delta$

ii) =-*formulas*: $\Sigma = \Delta$.

5.4.4. Additional concepts

1) *Contexts* (syntactic variables σ,ξ): *Any finite (possibly empty) sequence of* $\underline{E}$-*formulas* $x_i \mathrel{\underline{E}} \Sigma_i$, *separated by commas, where all* x_i *are different.*

2) Lines (syntactic variable λ)

   i) $\underline{EB}$-*lines* : $\sigma * x := \underline{EB} \mathrel{\underline{E}} \Sigma$

   ii) $\underline{PN}$-*lines* : $\sigma * p := \underline{PN} \mathrel{\underline{E}} \Sigma$

   iii) *Abbreviation lines*: $\sigma * d := \Delta \mathrel{\underline{E}} \Sigma$

3) *Books* (syntactic variable $\mathcal{B}$): *Any finite (possibly empty) sequence of lines, separated from one another by exclamation signs* (!).

5.4.5. Free variables

We define the *free variable* set $FV(\Sigma)$ of expressions Σ by induction on the structure of Σ (cf. section 5.4.2):

i) $FV(x) = \{x\}$

ii) $FV([x,\Gamma]\Delta) = FV(\Gamma) \cup (FV(\Delta)\backslash\{x\})$

iii) $FV(<\Gamma>\Delta) = FV(\Gamma) \cup FV(\Delta)$

iv) $FV(c(\Sigma_1,\ldots,\Sigma_k)) = \cup_{i=1,\ldots,k} FV(\Sigma_i)$

v) $FV(\underline{prop}) = FV(\underline{type}) = \emptyset$.

5.4.6. Substitution

1) The result of *simultaneous substitution* of $A_1,\ldots,A_k$ for the free variables $x_1,\ldots,x_k$ in an expression Σ is denoted by $[\![x_1,\ldots,x_k/A_1,\ldots,A_k]\!]\Sigma$ and locally abbreviated by Σ^*:

i) $x_i^* \equiv A_i$

ii) $y^* \equiv y$ if y not among $x_1,\ldots,x_k$

iii) $([y,\Sigma_1]\Sigma_2)^* = [y,\Sigma_1^*]\Sigma_2^*$ if y not among $x_1,\ldots,x_k$ and $x_i \in FV(\Sigma_2) \Rightarrow y \notin FV(A_i))$ for $i = 1,\ldots,k$ (otherwise rename y in $[y,\Sigma_1]\Sigma_2$).

iv) $(<\Sigma_1>\Sigma_2)^* \equiv <\Sigma_1^*>\Sigma_2^*$

v) $(c(\Sigma_1,\ldots,\Sigma_m))^* \equiv c(\Sigma_1^*,\ldots,\Sigma_m^*)$

vi) $\underline{prop}^* \equiv \underline{prop}$, $\underline{type}^* \equiv \underline{type}$.

2) *Substitution* of A for x is denoted by $[\![x/A]\!]$ and amounts to the case k = 1 above.

5.5. Correctness

5.5.1. Correct books

i) *the empty book is correct*

ii) *if* $\mathcal{B}$ *is correct and* λ *is correct with respect to* $\mathcal{B}$ *then* $\mathcal{B}!\lambda$ *correct.*

5.5.2. Correct context with respect to $\mathcal{B}$:

i) *the empty context is correct*

ii) *if* $\sigma * x := EB\ \underline{E}\ \Delta$ *is a line in the book* $\mathcal{B}$ *then* $\sigma, x\ \underline{E}\ \Delta$ *is a correct context with respect to* $\mathcal{B}$.

5.5.3. Correct lines with respect to $\mathcal{B}$:

1) $\underline{EB}$-lines: *If* $\mathcal{B}$; $\sigma \vdash^{(1)} \Delta$ *or* $\mathcal{B}$; $\sigma \vdash^{(2)} \Delta$, $\sigma \equiv x_1\ \underline{E}\ \Sigma_1, \ldots, x_k\ \underline{E}\ \Sigma_k$, *and* y *not among* $x_1, \ldots, x_k$ *then* $\sigma * y := \underline{EB}\ \underline{E}\ \Delta$ *is a correct line with respect to* $\mathcal{B}$.

2) $\underline{PN}$-lines: *If* $\mathcal{B}$; $\sigma \vdash^{(1)} \Delta$ *or* $\mathcal{B}$; $\sigma \vdash^{(2)} \Delta$ *and* p *does not occur in* $\mathcal{B}$ *then* $\sigma * p := \underline{PN}\ \underline{E}\ \Delta$ *is a correct line with respect to* $\mathcal{B}$.

3) Abbreviation lines: *If* $\mathcal{B}$;$\sigma \vdash \Sigma\ \underline{E}\ \Delta$ *and* d *does not occur in* $\mathcal{B}$ *then* $\sigma * d := \Sigma\ \underline{E}\ \Delta$ *is a correct line with respect to* $\mathcal{B}$.

5.5.4. Correct $\underline{E}$-formulas relative to a correct book $\mathcal{B}$ and a context σ which is correct w.r.t. $\mathcal{B}$

1) Repetition rule: *If* $\sigma \equiv x_1\ \underline{E}\ \Sigma_1, \ldots, x_k\ \underline{E}\ \Sigma_k$ *and* Σ_j *is an* i*-expression then* $\mathcal{B}$; $\sigma \vdash^{(i+1)} x_j\ \underline{E}\ \Sigma_j$ (for j = 1,...,k).

2) Abstraction rule: *If* $\mathcal{B}^* \equiv \mathcal{B}!\sigma * x := \underline{EB}\ \underline{E}\ \alpha$ *and* $\mathcal{B}^*$ *is correct and* $\mathcal{B}^*$; $\sigma, x\ \underline{E}\ \alpha \vdash^{(i)} \Sigma\ \underline{E}\ \Delta$ *then* $\mathcal{B}$; $\sigma \vdash^{(i)} [x,\alpha]\Sigma\ \underline{E}\ [x,\alpha]\Delta$.

3) Application rules:

   i) *If* $\vdash A\ \underline{E}\ \alpha$ *and* $\vdash^{(i)} B\ \underline{E}\ [x,\alpha]C$ *then* $\vdash^{(i)} \langle A\rangle B\ \underline{E}\ [\![x/A]\!]C$.

   ii) *If* $\vdash A\ \underline{E}\ \alpha$, $\vdash^{(i)} B\ \underline{E}\ C$ *and* $\vdash C\ \underline{E}\ [x,\alpha]D$ *then* $\vdash^{(i)} \langle A\rangle B\ \underline{E}\ \langle A\rangle C$ (clearly i will be 3 here).

4) Substitution rule: *If* Σ *is an* i*-expression and either* $x_1\ \underline{E}\ \Sigma_1, \ldots, x_k\ \underline{E}\ \Sigma_k * c := \underline{PN}\ \underline{E}\ \Sigma$ *or* $x_1\ \underline{E}\ \Sigma_1, \ldots, x_k\ \underline{E}\ \Sigma_k * c := \Delta\ \underline{E}\ \Sigma$ *is a line in the book* $\mathcal{B}$ *and* $\mathcal{B}$; $\sigma \vdash A_j\ \underline{E}\ [\![x_1, \ldots, x_k/A_1, \ldots, A_k]\!]\Sigma_j$ *for* j = 1,...,k *then* $\mathcal{B}$; $\sigma \vdash^{(i+1)} c(A_1, \ldots, A_k)\ \underline{E}\ [\![x_1, \ldots, x_k/A_1, \ldots, A_k]\!]\Sigma$.

5) Rule of type-conversion: *If* $\vdash \Delta \underline{E} \Sigma$ *and* $\vdash \Sigma = \Gamma$ *then* $\vdash \Delta \underline{E} \Gamma$.

6) Rules of type- and prop-inclusion:

i) *If* $\vdash \Sigma \underline{E} [x_1,\alpha_1] \ldots [x_k,\alpha_k][y,\beta] \underline{\text{type}}$ *(possibly* $k = 0$*) then* $\vdash \Sigma \underline{E} [x_1,\alpha_1] \ldots [x_k,\alpha_k]\underline{\text{type}}$.

ii) *If* $\vdash \Sigma \underline{E} [x_1,\alpha_1] \ldots [x_k,\alpha_k][y,\beta]\underline{\text{prop}}$ *(possibly* $k = 0$*) then* $\vdash \Sigma \underline{E} [x_1,\alpha_1] \ldots [x_k,\alpha_k]\underline{\text{prop}}$.

5.5.5. Correct expressions with respect to $\mathcal{B}$ and σ

1) Correct 1-expressions:

i) *If* $\mathcal{B}$ *is correct and* σ *is correct with respect to* $\mathcal{B}$ *then* $\mathcal{B}; \sigma \vdash^{(1)} \underline{\text{type}}$ *and* $\mathcal{B}; \sigma \vdash^{(1)} \underline{\text{prop}}$.

ii) *If* $\mathcal{B}^* \equiv \mathcal{B}!\sigma * x := \underline{EB} \underline{E} \alpha$ *and* $\mathcal{B}^*; \sigma,x \underline{E} \alpha \vdash^{(1)} \Delta$ *then* $\mathcal{B};\sigma \vdash^{(1)} [x,\alpha]\Delta$.

2) Correct 2- and 3-expressions: *If* $\vdash^{(i)} \Sigma \underline{E} \Delta$ *then* $\vdash^{(i)} \Sigma$.

Remark: It is intended that $\mathcal{B}; \sigma \vdash A$ or $\mathcal{B}; \sigma \vdash \varphi$ only if $\mathcal{B}$ is correct and σ is correct with respect to $\mathcal{B}$. This condition is explicitly imposed in 5.5.4 and 5.5.5.1i) and propagated all through the definition.

5.5.6. Correct =-formulas with respect to $\mathcal{B}$ and σ

1) β-equality: *If* $\vdash \langle A\rangle[x,\alpha]B$ *and* $\vdash [\![x/A]\!]B$ *then* $\vdash \langle A\rangle[x,\alpha]B = [\![x/A]\!]B$.

2) η-equality: *If* $\vdash [x,B]\langle x\rangle C$, *and* $x \notin FV(C)$ *and* $\vdash C$ *then* $\vdash [x,B]\langle x\rangle C = C$.

3) δ-equality: *If* $x_1 \underline{E} \Sigma_1,\ldots,x_k \underline{E} \Sigma_k * d := \Delta \underline{E} \Sigma$ *is a line in* $\mathcal{B}$, *and* $\mathcal{B}; \sigma \vdash A_j \underline{E} [\![x_1,\ldots,x_k/A_1,\ldots,A_k]\!]\Sigma_j$ *for* $j = 1,\ldots,k$, *and* $\mathcal{B}; \sigma \vdash [\![x_1,\ldots,x_k/A_1,\ldots,A_k]\!]\Delta$ *then* $\mathcal{B}; \sigma \vdash d(A_1,\ldots,A_k) = [\![x_1,\ldots,x_k/A_1,\ldots,A_k]\!]\Delta$

4) Monotonicity rules:

i) *If* $\mathcal{B}^* \equiv \mathcal{B}!\sigma * x := \underline{EB} \underline{E} \alpha$ *and* $\mathcal{B}^*; \sigma,x \underline{E} \alpha \vdash B_1 = B_2$ *then* $\mathcal{B}; \sigma \vdash [x,\alpha]B_1 = [x,\alpha]B_2$.

ii) *If* $\vdash \alpha_1 = \alpha_2$, $\vdash [x,\alpha_1]B$, *and* $\vdash [x,\alpha_2]B$ *then* $\vdash [x,\alpha_1]B = [x,\alpha_2]B$.

iii) *If* $\vdash A_1 = B_1$, $\vdash A_2 = B_2$, $\vdash \langle A_1\rangle A_2$, *and* $\vdash \langle B_1\rangle B_2$ *then* $\vdash \langle A_1\rangle A_2 = \langle B_1\rangle B_2$.

iv) *If* $\vdash A_j = B_j$ (for $j = 1,\ldots,k$), *and* $\vdash c(A_1,\ldots,A_k)$, *and* $\vdash c(B_1,\ldots,B_k)$ *then* $\vdash c(A_1,\ldots,A_k) = c(B_1,\ldots,B_k)$.

5) Reflexivity, symmetry and transitivity rules

i) *If* $\vdash A$, $\vdash B$ *and* $A \equiv B$ *then* $\vdash A = B$

ii) *If* $\vdash A = B$ *then* $\vdash B = A$

iii) *If* $\vdash A = B$, *and* $\vdash B = C$ *then* $\vdash A = C$.

Remark: It is intended that $\mathcal{B}; \sigma \vdash A = B$ only if both $\mathcal{B}; \sigma \vdash A$ and $\mathcal{B}; \sigma \vdash B$. In most cases above, though sometimes unnecessary, such conditions have been explicitly stated. Where they have been omitted it will be immediate that they hold by some other conditions.

6. Some remarks on language theory

6.1. Decidability

The language theory is mainly concerned with the investigation of the basic system. A major aim is to prove the *decidability* of the AUTOMATH languages. That is, to prove the existence of an effective procedure which for any given text in a finite amount of time decides whether it is correct or not (in AUT-QE, say). The kernel of such a *checker* deals with the *verification* of correctness of expressions and formulas (both $\underline{E}$- and =-formulas), relative to a given book and context (which are assumed to be correct already).

In this section we shall sketch a certain checking procedure, closely related to the actually running verifying program of Zandleven (cf. [11]). We shall also roughly indicate the proof of correspondence between the proposed checking procedure and the language definition of the preceding section.

6.2. Reduction

6.2.1. In order to study the =-relation in more detail we introduce the *reduction relation* $\geq$, a partial order among the expressions. For an explanation of the suggestive dots in our definition we refer to section 2.12.4.

6.2.2. Definition:

1) *One-step* reduction (with respect to a book $\mathcal{B}$)

   i) one-step β-reduction: $\ldots\langle A\rangle[x,\alpha]C\ldots > \ldots[\![x/A]\!]C\ldots$

   ii) one-step η-reduction: *If* $x \notin FV(C)$ *then* $\ldots[x,\alpha]\langle x\rangle C\ldots > \ldots C\ldots$

   iii) one-step δ-reduction: *If* d *was introduced by an abbreviation line* $x_1 \underline{E} \alpha_1,\ldots,x_k \underline{E} \alpha_k * d := D \underline{E} \Sigma$ *in* $\mathcal{B}$ *then* $\ldots d(\Sigma_1,\ldots,\Sigma_k)\ldots > \ldots[\![x_1,\ldots,x_k/\Sigma_1,\ldots,\Sigma_k]\!]D\ldots$

   iv) also $>$ is allowed with any *combination* of the indices such as: *If* $A >_\beta B$ *or* $A >_\eta B$ *then* $A >_{\beta\eta} B$

   v) one-step reduction *in general*: *If* $A >_{\beta\eta\delta} B$ *then* $A > B$.

2) *Many-step* reduction (with respect to $\mathcal{B}$)

   i) *If* $A \equiv B$ *then* $A \geq B$

   If $A \geq B$ *and* $B > C$ *(with respect to* $\mathcal{B}$*) then* $A \geq C$.

   So $\geq$ is the reflexive and transitive closure of $>$. Likewise $\geq_{\beta\delta}$ denotes the reflexive and transitive closure of $>_{\beta\delta}$ etc. For $A \geq B$ we also write $B \leq A$.

3) i) Reduction sequence: A sequence $\Sigma_1, \Sigma_2, \ldots$ of expressions is called a *reduction sequence* of Σ_1 if for all i we have $\Sigma_i \equiv \Sigma_{i+1}$ or $\Sigma_i > \Sigma_{i+1}$.

ii) Proper reduction sequence: A *reduction sequence* $\Sigma_1, \Sigma_2, \ldots$ is called *proper* if for all i we have $\Sigma_i > \Sigma_{i+1}$.

6.2.3. Clearly the =- relation is the equivalence relation generated by the restriction of > to correct expressions. So we can conclude: $\vdash A = B$ *iff* $A \equiv C_1 \geq D_1 \leq C_2 \geq D_2 \leq \ldots \geq D_{k-1} \leq C_k \equiv B$ *(possibly* $k = 1$*), where all expressions in the respective reduction sequences are correct.*

6.2.4. As an example of a reduction sequence consider:

3alt $>_\delta$ <2>succfun $>_\delta$ <2>[x,nat]successor(x) $>_\beta$ successor(2) $>_\delta$ successor(successor(1)) (see section 2.16). So each reduction step seems to bring us closer to some possible "outcome". Here β- and δ-reduction amount to evaluation and η-reduction to a certain simplification of expressions.

6.3. The three problems: normalization, Church-Rosser and closure

6.3.1. It will appear that the decision procedure for equations (=-formulas) plays a central role in the checker. At first we state - in terms of the remark in section 6.2.4 - two important questions around reduction and definitional equality:

i) *(Normalization)* Do correct expressions always have a final outcome, i.e. do they always reduce to an expression which does not reduce further?

ii) *(Church-Rosser property)* Do definitionally equal expressions have a common outcome, i.e. an expression to which they both reduce?

A third central question concerns the so-called *closure property* (this term was introduced by R.P. Nederpelt in the introduction to [9]):

iii) Is the system closed under reductions, i.e. do correct expressions remain correct under reduction?

6.3.2. Normalization and strong normalization

Let us define

1) A is *normal* if no one-step reduction A > B can be applied.

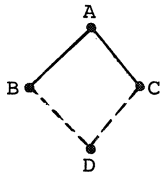
2) A is said to *normalize* if A reduces to some normal B (which is then called a *normal form* of A).

3) A is said to *strongly normalize* if all proper reduction sequences of A terminate.

We say that *normalization* (resp. *strong normalization)* holds if all correct expressions normalize (resp. strongly normalize). Normalization (and a fortiori strong normalization) does not hold in the full λ-calculus (take as a counter-example the expression <λx.<x>x>λx.<x>x). In typed systems such as AUTOMATH however, strong normalization (and hence normalization) does hold. Much work concerning (strong) normalization has been done by logicians studying systems of *natural deduction* and functional interpretations (cf. for instance [5], [8], [10]). Their methods often apply to AUTOMATH also. Some new proofs of normalization have been given by members of the AUTOMATH-project (cf. [9]).

6.3.3. Church-Rosser theorem; uniqueness of normal forms

Question 6.3.1ii) above amounts to the *Church-Rosser theorem: If* A = B *then* A ≥ C ≤ B *for some* C. An alternative formulation of this is the *Diamond property* for ≥: *If* A ≥ B *and* A ≥ C *then* B ≥ D ≤ C *for some* D (cf. figure).



Diamond property

As a corollary of the Church-Rosser theorem we mention the *uniqueness of normal forms: If* B *and* C *are normal forms of* A *then* B ≡ C. This property together with the normalization theorem allows us to speak of *the* normal form NF(A) - computable by an effective procedure NF - of correct expressions A. The Church-Rosser theorem holds in the full λ-calculus as well as in typed systems. In AUTOMATH languages without η-reduction the standard λ-calculus proofs simply carry over (cf. [9]). In fact, in view of strong normalization, a slightly easier proof can be given here. For, e.g., AUT-QE, where we have η-reduction the proof is somewhat more complicated and depends heavily on the closure theorem. The author intends to publish this proof and the other proofs omitted in this section in his doctoral dissertation.

6.3.4. Closure property

Let us first formulate the *closure theorem: If* $\mathcal{B}$; σ ⊢ A *(respectively* $\mathcal{B}$; σ ⊢ A $\underline{E}$ B) *and* A ≥ C *(with respect to* $\mathcal{B}$*) then* $\mathcal{B}$; σ ⊢ C *(respectively* $\mathcal{B}$; σ ⊢ C $\underline{E}$ B). In connection with the closure theorem, which holds for AUT-QE, we have two important derived rules:

1) *General substitution principle* (as mentioned in 2.5): *If* $x_1 \underline{E} \Sigma_1, \ldots, x_k \underline{E} \Sigma_k \vdash B$ (resp. $\vdash B \underline{E} C$) *and* $\sigma \vdash A_i \underline{E} \Sigma_i^*$ (for $i = 1, \ldots, k$) *then* $\sigma \vdash B^*$ (resp. $\vdash B^* \underline{E} C^*$), where Σ^* stands for $[\![x_1, \ldots, x_k / A_1, \ldots, A_k]\!]\Sigma$.
2) The *"left-hand equality rule"* (compare with the rule of type-conversion, which is the "right-hand equality rule"):
If $\vdash^{(3)} A \underline{E} B$ *and* $\vdash A = C$ *then* $\vdash C \underline{E} B$.
For 2-expression A we only have a weaker version in view of type-inclusion: *If* $\vdash^{(2)} A \underline{E} B$ *and* $\vdash A = C$ *and* $\vdash^{(2)} C \underline{E} D$ *then* $\vdash C \underline{E} B$ *or* $\vdash A \underline{E} D$.

6.4. A decision procedure

6.4.1. Deciding =-formulas

Suppose A and B are correct expressions. The normal form procedure NF (section 6.3.2) easily yields a decision method for the equation A = B, namely A = B iff NF(A) ≡ NF(B). Often, however, it is not necessary to compute normal forms for deciding A = B. For example, when A and B have different degrees one can easily draw a negative conclusion. Or more important, it generally happens that a few well-chosen reduction steps in A or B will result in a non-normal common reduct. The choice of efficient reduction steps here is a matter of *strategy;* the termination of a procedure which successively applies reduction rules to A or B is anyhow guaranteed by the strong normalization property, no matter in what order the reduction steps are applied.

In order to prove the correspondence between decision procedure and language definition we must know that all the expressions in the reduction sequences from A and B to some common reduct are correct again. This is indeed the case by the closure theorem.

6.4.2. Deciding E-formulas and expressions

6.4.2.1. Assume $\mathcal{B}$ is a correct book and σ a correct context; we must define a decision procedure for the correctness of E-formulas and expressions. It will appear that this problem can be reduced to the decision problem for =-formulas (but for the straightforward task of checking the validity of the identifiers used).

6.4.2.2. Uniqueness of types

We know (by the rule of type conversion) that for all B' with $\vdash B = B'$

we have $\vdash A \underline{E} B \Leftrightarrow \vdash A \underline{E} B'$.
For 3-expressions A the converse *(uniqueness of types*)* holds too:

(*) $\quad \vdash A \underline{E} B$ *and* $\vdash A \underline{E} B' \Rightarrow \vdash B = B'$.

For 2-expressions A we must be somewhat more precise in view of type-inclusion. We define among the correct expressions the relation $\underline{\subseteq}$ by:

i) $[x_1,\alpha_1]\ldots[x_k,\alpha_k][y,\beta]\underline{type} \subset [x_1,\alpha_1]\ldots[x_k,\alpha_k]\underline{type}$
ii) $[x_1,\alpha_1]\ldots[x_k,\alpha_k][y,\beta]\underline{prop} \subset [x_1,\alpha_1]\ldots[x_k,\alpha_k]\underline{prop}$
iii) $\underline{\subseteq}$ is the transitive closure of = and $\subset$.

Then instead of (*) for 2-expressions A we can prove

$$\vdash {}^{(2)}A \underline{E} B \text{ and } \vdash {}^{(2)}A \underline{E} B' \Rightarrow \vdash B \underline{\subseteq} B' \text{ or } \vdash B' \underline{\subseteq} B.$$

6.4.2.3. Now assume that A is correct. Then we can define a *"mechanical type" function* CAT, such that:

i) $\vdash {}^{(3)}A \underline{E} B \Leftrightarrow \vdash {}^{(3)}A$, $\vdash B$ and $\vdash CAT(A) = B$

ii) $\vdash {}^{(2)}A \underline{E} B \Leftrightarrow \vdash {}^{(2)}A$, $\vdash B$ and $\vdash CAT(A) \underline{\subseteq} B$.

So CAT computes some canonical representative of the class of B' with $\vdash A \underline{E} B'$; furthermore, this B' is minimal with respect to $\underline{\subseteq}$. For the actual definition of CAT we refer to [11, section 7]. Since the decision procedure $\overset{D}{=}$ for equations in the current checker also contains the possibility of type-inclusion - i.e. $A \overset{D}{=} B$ iff $A \underline{\subseteq} B$ - the type function CAT reduces the verification of $\underline{E}$-formulas to the verification of equations.

6.4.2.4. Finally we point out a decision procedure for correctness of expressions. Here we proceed by induction on the length of expressions. As an example we treat the case of application expressions <A>B where A and B are already supposed to be correct.

6.4.2.5. Uniqueness of domains

For function-like expressions A we define α to be the *domain* of A if

---

*) Here we mean uniqueness with respect to definitional equality (=), in contrast with section 6.3.3, where we mean uniqueness with respect to syntactic equality ($\equiv$).

$$\vdash A \underline{E} [x,\alpha]\Sigma \qquad \textit{or} \qquad \vdash {}^{(1)}A = [x,\alpha]\Sigma \ .$$

For domains we have *uniqueness* also (by the closure theorem and the Church-Rosser theorem): *If* α *and* β *are domains of* A *then* $\alpha = \beta$. This fact allows us to speak about *the* domain of function-like expressions. Now we are able to define a *"mechanical domain" function* DOM (for which we refer to [11, section 7]), which for function-like A picks out a canonical representative of the domain of A. The termination of DOM(A) follows by induction on the degree of A, using strong normalization.

6.4.2.6. By CAT and DOM the verification of correctness of <A>B reduces to the verification of some suitable equation: $\vdash$<A>B $\Leftrightarrow$ $\vdash$ A *and* $\vdash$ B *and* $\vdash$ A $\underline{E}$ DOM(B) or, equivalently, by 6.4.2.3i),

$$\vdash \text{<A>B} \Leftrightarrow \vdash A \textit{ and } \vdash B \textit{ and } \vdash \text{CAT(A)} = \text{DOM(B)} \ .$$

6.4.2.7. For the other cases of correctness of expressions we refer to Zandleven again. The correspondence of the current verifier with the actual language definition is either immediate or follows from the above facts about CAT and DOM.

7. References

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

[2] De Bruijn, N.G; *Automath, a language for mathematics*. Notes (prepared by B. Fawcett) of a series of lectures in the Séminaire de Mathématiques Supérieures, Université de Montréal, 1971.

[3] De Bruijn, N.G.; *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem*. Indag. Math., 34, no. 5. 1972.

[4] De Bruijn, N.G.; *The AUTOMATH Mathematics Checking Project*. This volume*).

[5] Girard, J.Y.; *Interpretation fonctionelle et élimination des coupures de l'arithmetique d'ordre supérieur*. Doctoral dissertation, Université Paris VII, 1972.

[6] Howard, W.A.; *The formulae-as-types notion of construction*. Unpublished 1969.

[7] Jutting, L.S. van Benthem; *The development of a text in AUT-QE*. This volume*).

[8] Martin-Löf, P.; *An intuionistic theory of types*. Unpublished 1972.

[9] Nederpelt, R.P.; *Strong normalization in a typed lambda-calculus with lambda-structured types*. Doctoral dissertation, Technological University, Eindhoven, 1972.

[10] Prawitz, D.; *Ideas and results in proof theory*. In: Proc. 2nd. Scandinavian Logic Symp. North-Holland Publ. Comp., Amsterdam, 1971.

[11] Zandleven, I.; *Verifying program for AUTOMATH*. This volume*).

*) "This volume" refers to: Proceedings of the Symposium on APL (Paris, December 1973), ed. P. Braffort.

Appendix 2. The paragraph system

In the definition of AUT-QE ([vD, 5]) it is required that constants which are identifier parts of different lines are different. In this appendix we describe a variant of AUTOMATH languages in which this rule is weakened. The AUT-QE version of this variant has actually been used for translating Landau's book. It is irrelevant for the following discussion, which particular AUTOMATH language is considered. We shall therefore presuppose an unspecified language AUT, and we shall call its paragraphed variant AUT-PAR.

1. Paragraph lines

A book in AUT-PAR can be split up into *paragraphs*. In the language we have three special symbols +, — and -, and a countable set of *paragraph identifiers* (which we shall denote here by syntactic variables s,s1,s2,...,t,t1,t2,...). There is a basic *paragraph identifier* cover . This will play the rôle of the empty environment; the word "cover" is meant to suggest "bookjacket". Besides ordinary AUTOMATH lines (which we will call here *proper lines*), we have a special sort of lines (called *paragraph lines*), which are used to indicate the paragraphs. There are two kinds of paragraph lines: *opening lines* which have the form +s, and *closing lines* which consist of the single symbol —.

2. The first rule for paragraph lines

For this description we shall number the lines of our book (proper lines as well as paragraph lines) in their proper order, and we will indicate lines by their numbers. For each line n we define o(n) (c(n) respectively) to be the number of opening lines (closing lines respectively) preceding it.

The first rule for paragraph lines is:

$$o(n) \geq c(n) \quad \text{for all } n.$$

It follows that the paragraph lines provide the book with a kind of nested structure.

The *paragraph level* of a line n is defined by $pl(n) = o(n) - c(n)$. For a line n with $pl(n) > 0$ we define its *paragraph opening* by $po(n) = \max\{m \mid m < n \text{ and } pl(m) < pl(n)\}$. It is easy to see that $pl(1) = 0$, that for each n with $pl(n) > 0$ the line po(n) is an opening line, and that $pl(po(n)) = pl(n) - 1$.

3. An example

As an example we represent schematically a book with paragraphs. The numbering of the lines in the book appears to the left. It only serves our (metalingual) discussion, and does not belong to the schematically indicated AUTOMATH text. The proper lines are indicated by their identifiers (constants or variables) and their contexts. The dots indicate middle parts and category parts.

```
1          * a := .....
2          * b := .....
3     +s
4          * x := ——— E .....
5        x * a := .....
6          * x := ——— E .....
7     +t
8        x * c := .....
9          * a := .....
10    —
11    —
12         * c := .....
13    +t
14         * c := .....
15    —
16    +s
17    +t
18         * x := ——— E .....
19       x * d := .....
20    —
```

In this example we have indeed $o(n) \geq c(n)$ for all n, and e.g.

$o(4) = 1$, $c(4) = 0$ hence $pl(4) = 1$

$o(16) = 3$, $c(16) = 3$ hence $pl(16) = 0$

$po(4) = 3$

$po(15) = 13$

$po(20) = 17$.

4. Indices and paragraphs

For references to paragraphs we use *indices*. An index has the form $s_1 - s_2 -...- s_u$ (with $u \geq 1$). The sign - is used as a separator, and has nothing to do with the — of closing lines. As a syntactic variable for indices we use $\bar{s}$. If $\bar{s} \equiv s_1 - s_2 -...- s_u$ then $\bar{s} - s$ denotes $s_1 - s_2 -...- s_u - s$. For each line n we define an index ind(n) as follows:

if pl(n) = 0 then ind(n) = cover.

if pl(n) > 0 and po(n) $\equiv$ +s then ind(n) = ind(po(n)) - s.

Note that, by this definition, for each n the first paragraph identifier in ind(n) is cover . Indices of the form cover-$s_2 -...- s_u$ are called *complete indices*. So, for all n, ind(n) is complete.

In the example we have:

ind(3) = cover
ind(4) = cover - s
ind(9) = cover - s - t
ind(15) = cover - t .

Given a book $\mathcal{B}$ and an index $\bar{s}$, the subsequence of $\mathcal{B}$ consisting of those lines n for which ind(n) = $\bar{s}$ is called the *paragraph* of $\bar{s}$. Note that paragraphs are mutually disjoint.

In our example the paragraphs are:

for $\bar{s} \equiv$ cover : 1,2,3,12,13,16
for $\bar{s} \equiv$ cover - s : 4,5,6,7,11,17
for $\bar{s} \equiv$ cover - s - t : 8,9,10,18,19,20
for $\bar{s} \equiv$ cover - t : 14,15.

If n is a line in a paragraph, and pl(n) > 0 then the line po(n) is called an *opener* of the paragraph. Note that the openers of a paragraph are *not* lines of that paragraph. The first opener of a paragraph is called the *paragraph opener* of that paragraph, the other openers are called *reopeners*. The closing lines in a paragraph are called the *closers* of that paragraph.

In our example we see:

for $\bar{s} \equiv$ cover - s the paragraph opener is 3, a reopener is 16 and a closer is 11.

for $\bar{s}$ = cover - s - t the paragraph opener is 7, a reopener is 17 and closers are 10 and 20.

for $\bar{s}$ = cover - t the paragraph opener is 13 and a closer is 15.

5. The rule for constants

The rule in AUTOMATH languages requiring that constants introduced in different lines are different, is weakened in the present language as follows:

Constants introduced in different proper lines in the same paragraph must be different.

Note that in our example this rule is observed.

For reference to a constant c introduced in the line n we use the constant *indexed*, i.e. we write c"$\bar{s}$" where $\bar{s}$ = ind(n). Note that for an indexed constant c"$\bar{s}$" the index $\bar{s}$ is always complete. In our example the constants a introduced in the lines 1, 5 and 9 appear indexed as a"cover" , a"cover - s" and a"cover - s - t" respectively.

By the rule for constants the indexed forms of constants introduced in different lines are different. So if we would replace each constant by its indexed form we would get a book where the strict rule for constants is observed.

6. The second rule for paragraph lines

It is an essential feature of our language that indices in indexed constants can be *abbreviated*, even (in some cases) to the point of omitting them entirely. For this purpose there is a second rule for paragraph lines:

If +s is a line with number n, then s may not occur in ind(n).

It follows that the paragraph identifiers of ind(n) are mutually different.

We shall now describe the interpretation of a constant c with abbreviated (or without) index. We assume that such a constant occurs in the middle part or category part of a proper line n with ind(n) = $\bar{s} = s_1 - s_2 - \ldots - s_k$. We distinguish three cases for the form of the abbreviated index.

i) c"$t_1 - t_2 - \ldots - t_\ell$" where $t_1 \not\equiv$ cover. In this case t_1 must be one (and therefore, by the second rule, exactly one) of $s_1, s_2, \ldots, s_k$. Suppose $t_1 \equiv s_i$ then c"$t_1 - t_2 - \ldots - t_\ell$" should be interpreted as c"$s_1 - s_2 - \ldots - s_i - t_2 - \ldots - t_\ell$". In our example, if a"s" occurs in the dots of line 19, it should be interpreted as a"cover - s".

ii) c"$- t_1 - t_2 - \ldots - t_\ell$" should be interpreted as c"$s_1 - s_2 - \ldots - s_k - t_1 - t_2 - \ldots - t_\ell$".
In our example, in the dots of line 12 a" - s" should be interpreted as a"cover - s" and a" - s - t" as a"cover - s - t".

iii) c appears without index. Then c should be interpreted as c"$\bar{t}$" where $\bar{t}$ is the "longest possible initial part of $\bar{s}$". I.e.: If c is identifier of a line preceding n in the paragraph of $\bar{s}$ then c should be interpreted as c"$\bar{s}$", else if c is identifier of a line preceding n in the paragraph of $s_1 - s_2 - \ldots - s_{k-1}$ then c should be interpreted as c"$s_1 - s_2 - \ldots - s_{k-1}$" etc.
In our example, in the dots of line 4, a should be interpreted as a"cover" , while in line 6 a should be interpreted as a"cover - s" . Note that in the middle part or category part of line 9 a should again be interpreted as a"cover - s" (i.e. the identifier introduced in line 5).

We see that the interpretation of a constant with abbreviated index depends on the place in the book where it occurs.

7. Reference to variables

According to the definition of AUT-QE, variables $x_1,\ldots,x_k$ of a context $x_1 \underline{E} \alpha_1,\ldots,x_k \underline{E} \alpha_k$ must be mutually different identifiers. We maintain this rule in AUT-PAR. Thus free variables occurring in the middle part or category part of a line always refer to a (unique) variable of the context (Cf. [vD, 2.4, 2.13.2, 5.5.2, 5.5.3]). Therefore such variables are never indexed.

For variables there are in AUTOMATH no restrictions to their use as identifier parts of different lines. If a variable x appears as a *context indicator* ([vD, 2.13.1 i)]) of a line n it always refers to the latest EB-line introducing x which precedes n. In AUT-PAR a context indicator must be indexed and for the indexed variables we allow the same abbreviation rules as in section 6. Hence the context indicator X in line 5 of our example should be interpreted (according to 6 iii) above) as X"cover - s" , i.e. the variable introduced in line 5. The context indicator X in line 8 should also be interpreted as X"cover - s" , but it refers to the variable introduced in line 6. In fact in lines n with n > 6 there is no possibility to use the variable X introduced in line 4 as a context indicator. The context indicator X in line 19 should be interpreted as X"cover - s - t" , thus referring to the variable introduced in line 18.

If we want to write line 19 on the context introduced in line 6 we should write:

```
19  X"s" * d := .....
```

or, with a complete index:

```
19  x"cover - s" * d := .....
```

It is allowed to introduce a new variable in line 19 by

```
x"s" * y := ——E ....
```

However

```
x"s" * x := ——E ....
```

would not be allowed, because this would give two variables X in one context.

8. Remarks on notation

Deviating from the notations for paragraph lines described above, we denote reopeners of a paragraph not by +s but by +*s, and closers of the paragraph of $s_1 - s_2 - \ldots - s_k$ by $-s_k$. Thus the lines 16 and 17 in our example should be written +*s and +*t , and the lines 11 and 15 as -s and -t respectively. This redundant notation is preferred for the sake of readability.

Appendix 3. The PN-lines from the preliminaries

```
   LAYOUT FROM FILE : EXCERPTOUTPUT/PREPNS     JANUARY 25, 1977    10:48:41


   +L

          * A          :=       ---                          ; PROP
        A * B          :=       ---                          ; PROP
        B * IMP        := [X,A]B                             ; PROP
1         * CON        :=       PN                           ; PROP
        A * NOT        := IMP(CON)                           ; PROP
        A * WEL        := NOT(NOT(A))                        ; PROP
        A * W          :=       ---                          ; WEL(A)
2       W * ET         :=       PN                           ; A
        B * EC         := IMP(A,NOT(B))                      ; PROP
        B * AND        := NOT(EC(A,B))                       ; PROP
          * SIGMA      :=       ---                          ; TYPE
    SIGMA * P          :=       ---                          ; [X,SIGMA]PROP
        P * ALL        := P                                  ; PROP
        P * NON        := [X,SIGMA]NOT(<X>P)                 ; [X,SIGMA]PROP
        P * SOME       := NOT(NON(P))                        ; PROP

   +E

    SIGMA * S          :=       ---                          ; SIGMA
        S * T          :=       ---                          ; SIGMA
3       T * IS         :=       PN                           ; PROP
4       S * REFIS      :=       PN                           ; IS(S,S)
        P * S          :=       ---                          ; SIGMA
        S * T          :=       ---                          ; SIGMA
        T * SP         :=       ---                          ; <S>P
       SP * I          :=       ---                          ; IS(S,T)
5       I * ISP        :=       PN                           ; <T>P
        P * AMONE      := [X,SIGMA]EY,SIGMA]EU,<X>P]EV,
                          <Y>P]IS(X,Y)                       ; PROP
        P * ONE        := AND(AMONE(SIGMA,P),
                          SOME(SIGMA,P))                     ; PROP
        P * O1         :=       ---                          ; ONE(SIGMA,P)
6      O1 * IND        :=       PN                           ; SIGMA
7      O1 * ONEAX      :=       PN                           ; <IND>P
    SIGMA * TAU        :=       ---                          ; TYPE
      TAU * F          :=       ---                          ; [X,SIGMA]TAU
        F * INJECTIVE  := ALL([X,SIGMA]ALL([Y,SIGMA]
                          IMP(IS(TAU,<X>F,<Y>F),IS(X,Y))
                          ))                                 ; PROP
        F * TO         :=       ---                          ; TAU
       TO * IMAGE      := SOME([X,SIGMA]IS(TAU,TO,<X>F))    ; PROP
      TAU * F          :=       ---                          ; [X,SIGMA]TAU
        F * G          :=       ---                          ; [X,SIGMA]TAU
        G * I          :=       ---                          ; [X,SIGMA]IS(TAU,<X>F,<X>G)
8       I * FISI       :=       PN                           ; IS([X,SIGMA]TAU,F,G)
9       P * OT         :=       PN                           ; TYPE
        P * O1         :=       ---                          ; OT
10     O1 * IN         :=       PN                           ; SIGMA
11     O1 * INP        :=       PN                           ; <IN>P
12      P * OTAX1      :=       PN                           ; INJECTIVE(OT,SIGMA,[X,OT]
                                                               IN(X))
        P * S          :=       ---                          ; SIGMA
        S * SP         :=       ---                          ; <S>P
13     SP * OTAX2      :=       PN                           ; IMAGE(OT,SIGMA,[X,OT]IN(X),S)
14    TAU * PAIRTYPE   :=       PN                           ; TYPE
      TAU * S          :=       ---                          ; SIGMA
        S * T          :=       ---                          ; TAU
15      T * PAIR       :=       PN                           ; PAIRTYPE
      TAU * P1         :=       ---                          ; PAIRTYPE
16     P1 * FIRST      :=       PN                           ; SIGMA
17     P1 * SECOND     :=       PN                           ; TAU
18     P1 * PAIRIS1    :=       PN                           ; IS(PAIRTYPE,PAIR(FIRST,
                                                               SECOND),P1)
19      T * FIRSTIS1   :=       PN                           ; IS(SIGMA,FIRST(PAIR),S)
20      T * SECONDIS1  :=       PN                           ; IS(TAU,SECOND(PAIR),T)

   -E
```

```
+*E

+ST

21  SIGMA * SET        :=      PN                          ; TYPE
    SIGMA * S          :=      ---                         ; SIGMA
        S * SO         :=      ---                         ; SET
22     SO * ESTI       :=      PN                          ; PROP
23      P * SETOF      :=      PN                          ; SET
        P * S          :=      ---                         ; SIGMA
        S * SP         :=      ---                         ; <S>P
24     SP * ESTII      :=      PN                          ; ESTI(S,SETOF(P))
        S * E          :=      ---                         ; ESTI(S,SETOF(P))
25      E * ESTIE      :=      PN                          ; <S>P
    SIGMA * SO         :=      ---                         ; SET
       SO * TO         :=      ---                         ; SET
       TO * INCL       := ALL([X,SIGMA]IMP(ESTI(X,SO),
                          ESTI(X,TO)))                     ; PROP
       TO * I          :=      ---                         ; INCL(SO,TO)
        I * J          :=      ---                         ; INCL(TO,SO)
26      J * ISSETI     :=      PN                          ; IS(SET,SO,TO)

-ST

-E

-L
```

Appendix 4. Excerpt for "Satz 27"

```
LAYOUT FROM FILE : EXCERPTOUTPUT/SATZ27     JANUARY 25, 1977    10:58:22


+L

      * A          :=       ---                          ; PROP
    A * B          :=       ---                          ; PROP
    B * IMP        := [X,A]B                             ; PROP
    B * A1         :=       ---                          ; A
   A1 * I          :=       ---                          ; IMP(A,B)
    I * MP         := <A1>I                              ; B
    B * C          :=       ---                          ; PROP
    C * I          :=       ---                          ; IMP(A,B)
    I * J          :=       ---                          ; IMP(B,C)
    J * TRIMP      := [X,A]<<X>I>J                       ; IMP(A,C)
      * CON        :=       PN                           ; PROP
    A * NOT        := IMP(CON)                           ; PROP
    A * WEL        := NOT(NOT(A))                        ; PROP
    A * A1         :=       ---                          ; A
   A1 * WELI       := [X,NOT(A)]<A1>X                    ; WEL(A)
    A * W          :=       ---                          ; WEL(A)
    W * ET         :=       PN                           ; A
    A * C1         :=       ---                          ; CON
   C1 * CONE       := ET([X,NOT(A)]C1)                   ; A

+IMP

    B * I          :=       ---                          ; IMP(A,B)
    I * J          :=       ---                          ; IMP(NOT(A),B)
    J * TH1        := ET(B,[X,NOT(B)]<<TRIMP(CON,I,
                      X)>J>X)                            ; B
    B * N          :=       ---                          ; NOT(A)
    N * TH2        := TRIMP(CON,B,N,[X,CON]CONE(B,X)
                      )                                  ; IMP(A,B)
    B * N          :=       ---                          ; NOT(B)
    N * I          :=       ---                          ; IMP(A,B)
    I * TH3        := TRIMP(CON,I,N)                     ; NOT(A)
    B * A1         :=       ---                          ; A
   A1 * N          :=       ---                          ; NOT(B)
    N * TH4        := [X,IMP(A,B)]<A1>TH3(N,X)           ; NOT(IMP(A,B))
    B * N          :=       ---                          ; NOT(IMP(A,B))
    N * TH5        := ET([X,NOT(A)]<TH2(X)>N)            ; A
    N * TH6        := [X,B]<[Y,A]X>N                     ; NOT(B)

-IMP

    B * EC         := IMP(A,NOT(B))                      ; PROP

+EC

    B * I          :=       ---                          ; IMP(A,NOT(B))
    I * TH1        := I                                  ; EC(A,B)
    B * I          :=       ---                          ; IMP(B,NOT(A))
    I * TH2        := [X,A][Y,B]<X><Y>I                  ; EC(A,B)

-EC

    B * E          :=       ---                          ; EC(A,B)
    E * A1         :=       ---                          ; A
   A1 * ECE1       := <A1>E                              ; NOT(B)
    E * B1         :=       ---                          ; B
   B1 * ECE2       := TH3"-IMP"(NOT(B),WELI(B,B1),E)     ; NOT(A)
```

```
     B * AND          := NOT(EC(A,B))                    ; PROP
     B * A1           :=      ---                        ; A
    A1 * B1           :=      ---                        ; B
    B1 * ANDI         := TH4"-IMP"(NOT(B),A1,WELI(B,B1)
                         )                               ; AND(A,B)
     B * A1           :=      ---                        ; AND(A,B)
    A1 * ANDE1        := TH5"-IMP"(NOT(B),A1)            ; A
    A1 * ANDE2        := ET(B,TH6"-IMP"(NOT(B),A1))      ; B

+AND

     B * N            :=      ---                        ; NOT(AND)
     N * A1           :=      ---                        ; A
    A1 * TH3          := ECE1(ET(EC,N),A1)               ; NOT(B)

-AND

     B * OR           := IMP(NOT(A),B)                   ; PROP
     B * A1           :=      ---                        ; A
    A1 * ORI1         := TH2"-IMP"(NOT(A),B,WELI(A1))    ; OR(A,B)
     B * B1           :=      ---                        ; B
    B1 * ORI2         := [X,NOT(A)]B1                    ; OR(A,B)

+OR

     B * I            :=      ---                        ; IMP(NOT(B),A)
     I * TH2          := [X,NOT]ET(B,TH3"L-IMP"(NOT(B),
                         A,X,I))                         ; OR(A,B)

-OR

     B * O            :=      ---                        ; OR(A,B)
     O * N            :=      ---                        ; NOT(A)
     N * ORE2         := <N>O                            ; B
     O * N            :=      ---                        ; NOT(B)
     N * ORE1         := ET(TH3"-IMP"(NOT(A),B,N,O))     ; A

+*OR

     B * N            :=      ---                        ; NOT(A)
     N * M            :=      ---                        ; NOT(B)
     M * TH3          := TH4"L-IMP"(NOT(A),B,N,M)        ; NOT(OR(A,B))

-OR

     C * O            :=      ---                        ; OR(A,B)
     O * I            :=      ---                        ; IMP(A,C)
     I * J            :=      ---                        ; IMP(B,C)
     J * ORAPP        := TH1"-IMP"(C,I,TRIMP(NOT,B,C,O,
                         J))                             ; C

+*OR

     O * I            :=      ---                        ; IMP(A,C)
     I * TH7          := TRIMP(NOT(C),NOT,B,[X,NOT(C)]
                         TH3"L-IMP"(A,C,X,I),O)          ; OR(C,B)
     O * I            :=      ---                        ; IMP(B,C)
     I * TH8          := TRIMP(NOT(A),B,C,O,I)           ; OR(A,C)
     C * D            :=      ---                        ; PROP
     D * O            :=      ---                        ; OR(A,B)
     O * I            :=      ---                        ; IMP(A,C)
     I * J            :=      ---                        ; IMP(B,D)
     J * TH9          := TH7(A,D,C,TH8(A,B,D,O,J),I)     ; OR(C,D)

-OR
```

```
      * SIGMA      :=      ---                          ; TYPE
SIGMA * P          :=      ---                          ; [X,SIGMA]PROP
    P * ALL        := P                                 ; PROP

+ALL

    P * S          :=      ---                          ; SIGMA
    S * N          :=      ---                          ; NOT(<S>P)
    N * TH1        := [X,ALL(SIGMA,P)]<<S>X>N           ; NOT(ALL(SIGMA,P))

-ALL

    P * NON        := [X,SIGMA]NOT(<X>P)                ; [X,SIGMA]PROP
    P * SOME       := NOT(NON(P))                       ; PROP
    P * S          :=      ---                          ; SIGMA
    S * SP         :=      ---                          ; <S>P
   SP * SOMEI      := TH1"-ALL"(NON(P),S,WELI(<S>P,
                      SP))                              ; SOME(SIGMA,P)

+SOME

    P * N          :=      ---                          ; NON(P)
    N * TH5        := WELI(NON(P),N)                    ; NOT(SOME(SIGMA,P))

-SOME

    P * S          :=      ---                          ; SOME(SIGMA,P)
    S * X          :=      ---                          ; PROP
    X * I          :=      ---                          ; [Y,SIGMA]IMP(<Y>P,X)

+*SOME

    I * N          :=      ---                          ; NOT(X)
    N * T          :=      ---                          ; SIGMA
    T * T5         := TH3"L-IMP"(<T>P,X,N,<T>I)         ; NOT(<T>P)
    N * T6         := MP(SOME(SIGMA,P),CON,S,
                      TH5([Y,SIGMA]T5(Y)))              ; CON

-SOME

    I * SOMEAPP    := ET(X,[Y,NOT(X)]T6"-SOME"(Y))      ; X

+*SOME

    P * Q          :=      ---                          ; [X,SIGMA]PROP
    Q * S          :=      ---                          ; SOME(SIGMA,P)
    S * I          :=      ---                          ; [X,SIGMA]IMP(<X>P,<X>Q)
    I * TH6        := SOMEAPP(S,SOME(Q),[X,SIGMA][Y,
                      <X>P]SOMEI(Q,X,MP(<X>P,<X>Q,Y,
                      <X>I)))                           ; SOME(Q)

-SOME

    C * AND3       := AND(A,AND(B,C))                   ; PROP
    C * A1         :=      ---                          ; AND3(A,B,C)
   A1 * AND3E1     := ANDE1(AND(B,C),A1)                ; A
   A1 * AND3E2     := ANDE1(B,C,ANDE2(AND(B,C),A1))     ; B
   A1 * AND3E3     := ANDE2(B,C,ANDE2(AND(B,C),A1))     ; C
    C * A1         :=      ---                          ; A
   A1 * B1         :=      ---                          ; B
   B1 * C1         :=      ---                          ; C
   C1 * AND3I      := ANDI(A,AND(B,C),A1,
                      ANDI(B,C,B1,C1))                  ; AND3(A,B,C)

+AND3

    C * A1         :=      ---                          ; AND3(A,B,C)
   A1 * TH1        := AND3I(B,C,A,AND3E2(A1),
                      AND3E3(A1),AND3E1(A1))            ; AND3(B,C,A)

-AND3
```

```
     C * EC3          := AND3(EC,EC(B,C),EC(C,A))          ; PROP
     C * E            :=      ---                          ; EC3(A,B,C)

+EC3

     E * TH1          := AND3E1(EC,EC(B,C),EC(C,A),E)      ; EC(A,B)
     E * TH3          := AND3E3(EC,EC(B,C),EC(C,A),E)      ; EC(C,A)
     E * TH4          := TH1'L-AND3'(EC,EC(B,C),EC(C,A)
                         ,E)                               ; EC3(B,C,A)

-EC3

     E * A1           :=      ---                          ; A
    A1 * EC3E12       := ECE1(TH1'-EC3',A1)                ; NOT(B)
    A1 * EC3E13       := ECE2(C,A,TH3'-EC3',A1)            ; NOT(C)
     E * B1           :=      ---                          ; B
    B1 * EC3E23       := EC3E12(B,C,A,TH4'-EC3',B1)        ; NOT(C)
    B1 * EC3E21       := EC3E13(B,C,A,TH4'-EC3',B1)        ; NOT(A)

+*EC3

     C * E            :=      ---                          ; EC(A,B)
     E * F            :=      ---                          ; EC(B,C)
     F * G            :=      ---                          ; EC(C,A)
     G * TH6          := AND3I(EC,EC(B,C),EC(C,A),E,F,
                         G)                                ; EC3(A,B,C)

-EC3

+E

 SIGMA * S            :=      ---                          ; SIGMA
     S * T            :=      ---                          ; SIGMA
     T * IS           :=      PN                           ; PROP
     S * REFIS        :=      PN                           ; IS(S,S)
     P * S            :=      ---                          ; SIGMA
     S * T            :=      ---                          ; SIGMA
     T * SP           :=      ---                          ; <S>P
    SP * I            :=      ---                          ; IS(S,T)
     I * ISP          :=      PN                           ; <T>P
 SIGMA * S            :=      ---                          ; SIGMA
     S * T            :=      ---                          ; SIGMA
     T * I            :=      ---                          ; IS(S,T)
     I * SYMIS        := ISP([X,SIGMA]IS(X,S),S,T,
                         REFIS(S),I)                       ; IS(T,S)
     T * U            :=      ---                          ; SIGMA
     U * I            :=      ---                          ; IS(S,T)
     I * J            :=      ---                          ; IS(T,U)
     J * TRIS         := ISP([X,SIGMA]IS(X,U),T,S,J,
                         SYMIS(I))                         ; IS(S,U)
     U * I            :=      ---                          ; IS(S,U)
     I * J            :=      ---                          ; IS(T,U)
     J * TRIS2        := TRIS(S,U,T,I,SYMIS(T,U,J))        ; IS(S,T)
     T * N            :=      ---                          ; NOT(IS(S,T))
     N * SYMNOTIS     := TH3'L-IMP'(IS(T,S),IS(S,T),N,
                         [X,IS(T,S)]SYMIS(T,S,X))          ; NOT(IS(T,S))

+NOTIS

     U * N            :=      ---                          ; NOT(IS(S,T))
     N * I            :=      ---                          ; IS(T,U)
     I * TH3          := ISP([X,SIGMA]NOT(IS(S,X)),T,U,
                         N,I)                              ; NOT(IS(S,U))
     N * I            :=      ---                          ; IS(U,T)
     I * TH4          := TH3(SYMIS(U,T,I))                 ; NOT(IS(S,U))

-NOTIS
```

```
     U * V          :=      ---                          ; SIGMA
     V * I          :=      ---                          ; IS(S,T)
     I * J          :=      ---                          ; IS(T,U)
     J * K          :=      ---                          ; IS(U,V)
     K * TR3IS      := TRIS(S,U,V,TRIS(I,J),K)           ; IS(S,V)
     V * W          :=      ---                          ; SIGMA
     W * I          :=      ---                          ; IS(S,T)
     I * J          :=      ---                          ; IS(T,U)
     J * K          :=      ---                          ; IS(U,V)
     K * L          :=      ---                          ; IS(V,W)
     L * TR4IS      := TRIS(S,V,W,TR3IS(I,J,K),L)        ; IS(S,W)
     P * AMONE      := [X,SIGMA][Y,SIGMA][U,<X>P][V,
                       <Y>P]IS(X,Y)                      ; PROP
     P * ONE        := AND(AMONE(SIGMA,P),
                       SOME(SIGMA,P))                    ; PROP
     P * A1         :=      ---                          ; AMONE(SIGMA,P)
    A1 * S          :=      ---                          ; SOME(SIGMA,P)
     S * ONEI       := ANDI(AMONE(SIGMA,P),
                       SOME(SIGMA,P),A1,S)               ; ONE(SIGMA,P)
     P * O1         :=      ---                          ; ONE(SIGMA,P)
    O1 * IND        :=      PN                           ; SIGMA
    O1 * ONEAX      :=      PN                           ; <IND>P
 SIGMA * TAU        :=      ---                          ; TYPE
   TAU * F          :=      ---                          ; [X,SIGMA]TAU
     F * S          :=      ---                          ; SIGMA
     S * T          :=      ---                          ; SIGMA
     T * I          :=      ---                          ; IS(S,T)
     I * ISF        := ISP(SIGMA,[X,SIGMA]IS(TAU,<S>
                       F,<X>F),S,T,REFIS(TAU,<S>F),I)    ; IS(TAU,<S>F,<T>F)
   TAU * F          :=      ---                          ; [X,SIGMA]TAU
     F * G          :=      ---                          ; [X,SIGMA]TAU
     G * I          :=      ---                          ; IS([X,SIGMA]TAU,F,G)
     I * S          :=      ---                          ; SIGMA
     S * FISE       := ISP([X,SIGMA]TAU,[Y,[X,SIGMA]
                       TAU]IS(TAU,<S>F,<S>Y),F,G,
                       REFIS(TAU,<S>F),I)                ; IS(TAU,<S>F,<S>G)
     G * I          :=      ---                          ; [X,SIGMA]IS(TAU,<X>F,<X>G)
     I * FISI       :=      PN                           ; IS([X,SIGMA]TAU,F,G)

-E

+*E

+ST

 SIGMA * SET        :=      PN                           ; TYPE
 SIGMA * S          :=      ---                          ; SIGMA
     S * SO         :=      ---                          ; SET
    SO * ESTI       :=      PN                           ; PROP
     P * SETOF      :=      PN                           ; SET
     P * S          :=      ---                          ; SIGMA
     S * SP         :=      ---                          ; <S>P
    SP * ESTII      :=      PN                           ; ESTI(S,SETOF(P))
     S * E          :=      ---                          ; ESTI(S,SETOF(P))
     E * ESTIE      :=      PN                           ; <S>P

+EQ
```

```
+LANDAU

+N

       * NAT          :=       PN                          # TYPE
       * X            :=       ---                         # NAT
     X * Y            :=       ---                         # NAT
     Y * IS           := IS'E'(NAT,X,Y)                    # PROP
     Y * NIS          := NOT(IS(X,Y))                      # PROP
     X * S            :=       ---                         # SET(NAT)
     S * IN           := ESTI(NAT,X,S)                     # PROP
       * P            :=       ---                         # [X,NAT]PROP
     P * SOME         := SOME'L'(NAT,P)                    # PROP
     P * ALL          := ALL'L'(NAT,P)                     # PROP
       * 1            :=       PN                          # NAT
       * SUC          :=       PN                          # [X,NAT]NAT
       * X            :=       ---                         # NAT
     X * Y            :=       ---                         # NAT
     Y * I            :=       ---                         # IS(X,Y)
     I * AX2          := ISF(NAT,NAT,SUC,X,Y,I)            # IS(<X>SUC,<Y>SUC)
       * AX3          :=       PN                          # [X,NAT]NIS(<X>SUC,1)
       * AX4          :=       PN                          # [X,NAT][Y,NAT][U,IS(<X>SUC,
                                                             <Y>SUC)]IS(X,Y)
       * S            :=       ---                         # SET(NAT)
     S * COND1        := IN(1,S)                           # PROP
     S * COND2        := ALL([X,NAT]IMP(IN(X,S),IN(<X>
                         SUC,S)))                          # PROP
       * AX5          :=       PN                          # [S,SET(NAT)][U,COND1(S)]
                                                             [V,COND2(S)][X,NAT]IN(X,S)
       * P            :=       ---                         # [X,NAT]PROP
     P * 1P           :=       ---                         # <1>P
    1P * XSP          :=       ---                         # [X,NAT][Y,<X>P]<<X>SUC>P
   XSP * X            :=       ---                         # NAT

+I1

     X * S            := SETOF(NAT,P)                      # SET(NAT)
     X * T1           := ESTII(NAT,P,1,1P)                 # COND1(S)
     X * Y            :=       ---                         # NAT
     Y * YES          :=       ---                         # IN(Y,S)
   YES * T2           := ESTIE(NAT,P,Y,YES)                # <Y>P
   YES * T3           := ESTII(NAT,P,<Y>SUC,<T2><Y>XSP)    # IN(<Y>SUC,S)
     X * T4           := <X><[Y,NAT][U,IN(Y,S)]T3(Y,U)>
                         <T1><S>AX5                        # IN(X,S)

-I1

     X * INDUCTION    := ESTIE(NAT,P,X,T4'-I1')            # <X>P
       * X            :=       ---                         # NAT
     X * Y            :=       ---                         # NAT
     Y * N            :=       ---                         # NIS(X,Y)

+21

     N * I            :=       ---                         # IS(<X>SUC,<Y>SUC)
     I * T1           := <I><Y><X>AX4                      # IS(X,Y)

-21

     N * SATZ1        := TH3'L-IMP'(IS(<X>SUC,<Y>SUC),
                         IS(X,Y),N,[U,IS(<X>SUC,<Y>SUC)
                         ]T1'-21'(U))                      # NIS(<X>SUC,<Y>SUC)
```

```
+23

     X * PROP1      := OR(IS(X,1),SOME([U,NAT]IS(X,
                       <U>SUC)))                    # PROP
       * T1         := ORI1(IS(1,1),SOME([U,NAT]IS(1,
                       <U>SUC)),REFIS(NAT,1))       # PROP1(1)
     X * T2         := SOMEI(NAT,[U,NAT]IS(<X>SUC,<U>
                       SUC),X,REFIS(NAT,<X>SUC))    # SOME([U,NAT]IS(<X>SUC,<U>SUC)
                                                       )
     X * T3         := ORI2(IS(<X>SUC,1),SOME([U,NAT]
                       IS(<X>SUC,<U>SUC)),T2)       # PROP1(<X>SUC)
     X * T4         := INDUCTION([Y,NAT]PROP1(Y),T1,
                       [Y,NAT][U,PROP1(Y)]T3(Y),X)  # PROP1(X)

-23

     X * N          :=      ---                     # NIS(X,1)
     N * SATZ3      := ORE2(IS(X,1),SOME([U,NAT]IS(X,
                       <U>SUC)),T4"-23",N)          # SOME([U,NAT]IS(X,<U>SUC))
     Y * Z          :=      ---                     # NAT

+24

     X * F          :=      ---                     # [Y,NAT]NAT
     F * PROP1      := ALL([Y,NAT]IS(<<Y>SUC>F,<<Y>F>
                       SUC))                        # PROP
     F * PROP2      := AND(IS(<1>F,<X>SUC),PROP1)   # PROP
     X * A          :=      ---                     # [Y,NAT]NAT
     A * B          :=      ---                     # [Y,NAT]NAT
     B * PA         :=      ---                     # PROP2(A)
    PA * PB         :=      ---                     # PROP2(B)
    PB * Y          :=      ---                     # NAT
     Y * PROP3      := IS(<Y>A,<Y>B)                # PROP
    PB * T1         := ANDE1(IS(<1>A,<X>SUC),PROP1(A)
                       ,PA)                         # IS(<1>A,<X>SUC)
    PB * T2         := ANDE1(IS(<1>B,<X>SUC),PROP1(B)
                       ,PB)                         # IS(<1>B,<X>SUC)
    PB * T3         := TRIS2(NAT,<1>A,<1>B,<X>SUC,T1,
                       T2)                          # PROP3(1)
     Y * P          :=      ---                     # PROP3(Y)
     P * T4         := AX2(<Y>A,<Y>B,P)             # IS(<<Y>A>SUC,<<Y>B>SUC)
     P * T5         := ANDE2(IS(<1>A,<X>SUC),PROP1(A)
                       ,PA)                         # PROP1(A)
     P * T6         := ANDE2(IS(<1>B,<X>SUC),PROP1(B)
                       ,PB)                         # PROP1(B)
     P * T7         := <Y>T5                        # IS(<<Y>SUC>A,<<Y>A>SUC)
     P * T8         := <Y>T6                        # IS(<<Y>SUC>B,<<Y>B>SUC)
     P * T9         := TR3IS(NAT,<<Y>SUC>A,<<Y>A>SUC,
                       <<Y>B>SUC,<<Y>SUC>B,T7,T4,
                       SYMIS"E"(NAT,<<Y>SUC>B,<<Y>B>
                       SUC,T8))                     # PROP3(<Y>SUC)
     Y * T10        := INDUCTION([Z,NAT]PROP3(Z),T3,
                       [Z,NAT][U,PROP3(Z)]T9(Z,U),Y) # PROP3(Y)
    PB * T11        := FISI(NAT,NAT,A,B,[Y,NAT]T10(Y)
                       )                            # IS"E"([Y,NAT]NAT,A,B)
     X * AA         := [Z,[Y,NAT]NAT][U,[Y,NAT]NAT]
                       [V,PROP2(Z)][W,PROP2(U)]T11(Z,
                       U,V,W)                       # AMONE([Y,NAT]NAT,[Z,[Y,NAT]
                                                       NAT]PROP2(Z))
     X * PROP4      := SOME"L"([Y,NAT]NAT,[Z,[Y,NAT]
                       NAT]PROP2(Z))                # PROP
       * T12        := [X,NAT]REFIS(NAT,<<X>SUC>SUC) # PROP1(1,SUC)
       * T13        := ANDI(IS(<1>SUC,<1>SUC),
                       PROP1(1,SUC),REFIS(NAT,<1>SUC)
                       ,T12)                        # PROP2(1,SUC)
       * T14        := SOMEI([Y,NAT]NAT,[Z,[Y,NAT]
                       NAT]PROP2(1,Z),SUC,T13)      # PROP4(1)
```

```
     X * P          :=      ---                          ; PROP4(X)
     P * F          :=      ---                          ; [Y,NAT]NAT
     F * PF         :=      ---                          ; PROP2(F)
    PF * G          := [Y,NAT]<<Y>F>SUC                  ; [Y,NAT]NAT
    PF * Y          :=      ---                          ; NAT
     Y * T15        := REFIS(NAT,<Y>G)                   ; IS(<Y>G,<<Y>F>SUC)
    PF * T16        := ANDE1(IS(<1>F,<X>SUC),PROP1(F)
                       ,PF)                              ; IS(<1>F,<X>SUC)
    PF * T17        := TRIS(NAT,<1>G,<<1>F>SUC,<<X>
                       SUC>SUC,T15(1),AX2(<1>F,<X>
                       SUC,T16))                         ; IS(<1>G,<<X>SUC>SUC)
     Y * T18        := ANDE2(IS(<1>F,<X>SUC),PROP1(F)
                       ,PF)                              ; PROP1(F)
     Y * T19        := <Y>T18                            ; IS(<<Y>SUC>F,<<Y>F>SUC)
     Y * T20        := TRIS2(NAT,<<Y>SUC>F,<Y>G,<<Y>
                       F>SUC,T19,T15)                    ; IS(<<Y>SUC>F,<Y>G)
     Y * T21        := TRIS(NAT,<<Y>SUC>G,<<<Y>SUC>F>
                       SUC,<<Y>G>SUC,T15(<Y>SUC),
                       AX2(<<Y>SUC>F,<Y>G,T20))          ; IS(<<Y>SUC>G,<<Y>G>SUC)
    PF * T22        := [Y,NAT]T21(Y)                     ; PROP1(<X>SUC,G)
    PF * T23        := ANDI(IS(<1>G,<<X>SUC>SUC),
                       PROP1(<X>SUC,G),T17,T22)          ; PROP2(<X>SUC,G)
    PF * T24        := SOMEI([Y,NAT]NAT,[Z,[Y,NAT]
                       NAT]PROP2(<X>SUC,Z),G,T23)        ; PROP4(<X>SUC)
     P * T25        := SOMEAPP([Y,NAT]NAT,[Z,[Y,NAT]
                       NAT]PROP2(Z),P,PROP4(<X>SUC),
                       [Z,[Y,NAT]NAT]CU,PROP2(Z)]
                       T24(Z,U))                         ; PROP4(<X>SUC)
     X * BB         := INDUCTION([Y,NAT]PROP4(Y),T14,
                       [Y,NAT]CU,PROP4(Y)]T25(Y,U),X)    ; PROP4(X)

-24

     X * SATZ4      := ONEI([Y,NAT]NAT,[Z,[Y,NAT]NAT]
                       PROP2"-24"(Z),AA"-24",BB"-24")    ; ONE"E"([Y,NAT]NAT,[Z,[Y,NAT]
                                                           NAT]AND(IS(<1>Z,<X>SUC),
                                                           ALL([Y,NAT]IS(<<Y>SUC>Z,<<Y>
                                                           Z>SUC))))
     X * PLUS       := IND([Y,NAT]NAT,[Z,[Y,NAT]NAT]
                       PROP2"-24"(Z),SATZ4)              ; [Y,NAT]NAT
     Y * PL         := <Y>PLUS                           ; NAT

+*24

     X * T26        := ONEAX([Y,NAT]NAT,[Z,[Y,NAT]
                       NAT]PROP2(Z),SATZ4)               ; PROP2(PLUS)

-24

     X * SATZ4A     := ANDE1(IS(<1>PLUS,<X>SUC),
                       PROP1"-24"(PLUS),T26"-24")        ; IS(PL(X,1),<X>SUC)

+*24

     X * T27        := ANDE2(IS(<1>PLUS,<X>SUC),
                       PROP1(PLUS),T26)                  ; PROP1(PLUS)

-24

     Y * SATZ4B     := <Y>T27"-24"                       ; IS(PL(X,<Y>SUC),<PL(X,Y)>SUC)

+*24

       * T28        := T11(1,PLUS(1),SUC,T26(1),T13)     ; IS"E"([Y,NAT]NAT,PLUS(1),SUC)

-24

     X * SATZ4C     := FISE(NAT,NAT,PLUS(1),SUC,
                       T28"-24",X)                       ; IS(PL(1,X),<X>SUC)
```

```
+*24

     X * T29        := T11(<X>SUC,PLUS(<X>SUC),
                       [Y,NAT]<<Y>PLUS>SUC,T26(<X>
                       SUC),T23(BB,PLUS,T26))          ; IS*E*([Y,NAT]NAT,PLUS(<X>SUC)
                                                         ,[Y,NAT]<<Y>PLUS>SUC)

-24

     Y * SATZ4D     := FISE(NAT,NAT,PLUS(<X>SUC),
                       [Z,NAT]<<Z>PLUS>SUC,T29*-24*,
                       Y)                              ; IS(PL(<X>SUC,Y),<PL(X,Y)>SUC)
     X * SATZ4E     := SYMIS(NAT,PL(X,1),<X>SUC,
                       SATZ4A)                         ; IS(<X>SUC,PL(X,1))
     Y * SATZ4F     := SYMIS(NAT,PL(X,<Y>SUC),<PL
                       (X,Y)>SUC,SATZ4B)               ; IS(<PL(X,Y)>SUC,PL(X,<Y>SUC))
     X * SATZ4G     := SYMIS(NAT,PL(1,X),<X>SUC,
                       SATZ4C)                         ; IS(<X>SUC,PL(1,X))
     Z * I          :=      ---                        ; IS(X,Y)
     I * ISPL1      := ISF(NAT,NAT,[U,NAT]PL(U,Z),X,
                       Y,I)                            ; IS(PL(X,Z),PL(Y,Z))
     I * ISPL2      := ISF(NAT,NAT,[U,NAT]PL(Z,U),X,
                       Y,I)                            ; IS(PL(Z,X),PL(Z,Y))

+25

     Z * PROP1      := IS(PL(PL(X,Y),Z),PL(X,PL(Y,Z))
                       )                               ; PROP
     Y * T1         := TR3IS(NAT,PL(PL(X,Y),1),<PL
                       (X,Y)>SUC,PL(X,<Y>SUC),
                       PL(X,PL(Y,1)),SATZ4A(PL(X,Y)),
                       SATZ4F,ISPL2(<Y>SUC,PL(Y,1),X,
                       SATZ4E(Y)))                     ; PROP1(1)
     Z * P          :=      ---                        ; PROP1(Z)
     P * T2         := AX2(PL(PL(X,Y),Z),PL(X,PL(Y,Z)
                       ),P)                            ; IS(<PL(PL(X,Y),Z)>SUC,<PL
                                                         (X,PL(Y,Z))>SUC)
     P * T3         := TR4IS(NAT,PL(PL(X,Y),<Z>SUC),
                       <PL(PL(X,Y),Z)>SUC,<PL(X,PL
                       (Y,Z))>SUC,PL(X,<PL(Y,Z)>SUC),
                       PL(X,PL(Y,<Z>SUC)),
                       SATZ4B(PL(X,Y),Z),T2,
                       SATZ4F(X,PL(Y,Z)),
                       ISPL2(<PL(Y,Z)>SUC,PL(Y,<Z>
                       SUC),X,SATZ4F(Y,Z)))            ; PROP1(<Z>SUC)

-25

     Z * SATZ5      := INDUCTION([U,NAT]PROP1*-25*(U)
                       ,T1*-25*,[U,NAT][V,PROP1*-25*
                       (U)]T3*-25*(U,V),Z)             ; IS(PL(PL(X,Y),Z),PL(X,PL(Y,Z)
                                                         ))
     Z * ASSPL1     := SATZ5                           ; IS(PL(PL(X,Y),Z),PL(X,PL(Y,Z)
                                                         ))

+26

     Y * PROP1      := IS(PL(X,Y),PL(Y,X))             ; PROP
     Y * T1         := SATZ4A(Y)                       ; IS(PL(Y,1),<Y>SUC)
     Y * T2         := SATZ4C(Y)                       ; IS(PL(1,Y),<Y>SUC)
     Y * T3         := TRIS2(NAT,PL(1,Y),PL(Y,1),<Y>
                       SUC,T2,T1)                      ; PROP1(1,Y)
     Y * P          :=      ---                        ; PROP1(X,Y)
     P * T4         := TRIS(NAT,<PL(X,Y)>SUC,<PL(Y,X)
                       >SUC,PL(Y,<X>SUC),AX2(PL(X,Y),
                       PL(Y,X),P),SATZ4F(Y,X))         ; IS(<PL(X,Y)>SUC,PL(Y,<X>SUC))
     P * T5         := SATZ4D                          ; IS(PL(<X>SUC,Y),<PL(X,Y)>SUC)
     P * T6         := TRIS(NAT,PL(<X>SUC,Y),<PL(X,Y)
                       >SUC,PL(Y,<X>SUC),T5,T4)        ; PROP1(<X>SUC,Y)

-26

     Y * SATZ6      := INDUCTION([Z,NAT]PROP1*-26*(Z,
                       Y),T3*-26*,[Z,NAT]
                       [U,PROP1*-26*(Z,Y)]T6*-26*(Z,
                       Y,U),X)                         ; IS(PL(X,Y),PL(Y,X))
     Y * COMPL      := SATZ6                           ; IS(PL(X,Y),PL(Y,X))
```

```
+27

     Y * PROP1       := NIS(Y,PL(X,Y))                    ; PROP
     X * T1          := SYMNOTIS(NAT,<X>SUC,1,<X>AX3)     ; NIS(1,<X>SUC)
     X * T2          := TH4'E-NOTIS'(NAT,1,<X>SUC,
                        PL(X,1),T1,SATZ4A)                ; PROP1(1)
     Y * P           :=      ---                          ; PROP1(Y)
     P * T3          := SATZ1(Y,PL(X,Y),P)                ; NIS(<Y>SUC,<PL(X,Y)>SUC)
     P * T4          := TH4'E-NOTIS'(NAT,<Y>SUC,<PL
                        (X,Y)>SUC,PL(X,<Y>SUC),T3,
                        SATZ4B)                           ; PROP1(<Y>SUC)

-27

     Y * SATZ7       := INDUCTION([Z,NAT]PROP1'-27'(Z)
                        ,T2'-27',[Z,NAT][U,PROP1'-27'
                        (Z)]T4'-27'(Z,U),Y)               ; NIS(Y,PL(X,Y))
     Z * DIFFPROP    := IS(X,PL(Y,Z))                     ; PROP

+29

     Y * I           := IS(X,Y)                           ; PROP
     Y * II          := SOME([U,NAT]DIFFPROP(X,Y,U))      ; PROP
     Y * III         := SOME([V,NAT]DIFFPROP(Y,X,V))      ; PROP
     Y * ONE1        :=      ---                          ; I
   ONE1 * U          :=      ---                          ; NAT
     U * T1          := TRIS(NAT,PL(U,X),PL(X,U),
                        PL(Y,U),COMPL(U,X),
                        ISPL1(U,ONE1))                    ; IS(PL(U,X),PL(Y,U))
     U * T2          := TH3'E-NOTIS'(NAT,X,PL(U,X),
                        PL(Y,U),SATZ7(U,X),T1)            ; NIS(X,PL(Y,U))
   ONE1 * T3         := TH5'L-SOME'(NAT,[U,NAT]
                        DIFFPROP(U),[U,NAT]T2(U))         ; NOT(II)
     Y * T4          := TH1'L-EC'(I,II,[Z,I]T3(Z))        ; EC(I,II)
   ONE1 * T5         := T3(Y,X,SYMIS(NAT,X,Y,ONE1))       ; NOT(III)
     Y * T6          := TH2'L-EC'(III,I,[Z,I]T5(Z))       ; EC(III,I)
     Y * TWO1        :=      ---                          ; II
   TWO1 * THREE1     :=      ---                          ; III
 THREE1 * U          :=      ---                          ; NAT
     U * DU          :=      ---                          ; DIFFPROP(X,Y,U)
    DU * V           :=      ---                          ; NAT
     V * DV          :=      ---                          ; DIFFPROP(Y,X,V)
    DV * T6A         := TR4IS(NAT,X,PL(Y,U),PL(PL(X,V)
                        ,U),PL(X,PL(V,U)),PL(PL(V,U),
                        X),DU,ISPL1(Y,PL(X,V),U,DV),
                        ASSPL1(X,V,U),COMPL(X,PL(V,U))
                        )                                 ; IS(X,PL(PL(V,U),X))
    DV * T7          := MP(IS(X,PL(PL(V,U),X)),CON,
                        T6A,SATZ7(PL(V,U),X))             ; CON
    DU * T8          := SOMEAPP(NAT,[V,NAT]DIFFPROP(Y,
                        X,V),THREE1,CON,[V,NAT]
                        [DV,DIFFPROP(Y,X,V)]T7(V,DV))     ; CON
 THREE1 * T9         := SOMEAPP(NAT,[U,NAT]DIFFPROP(U)
                        ,TWO1,CON,[U,NAT][DU,DIFFPROP
                        (U)]T8(U,DU))                     ; CON
   TWO1 * T10        := [Z,III]T9(Z)                      ; NOT(III)
     Y * T11         := TH1'L-EC'(II,III,[Z,II]T10(Z))    ; EC(II,III)
     Y * A           := TH6'L-EC3'(I,II,III,T4,T11,T6)    ; EC3(I,II,III)

-29

     Y * SATZ9B      := A'-29'                            ; EC3(IS(X,Y),SOME([U,NAT]
                                                            DIFFPROP(X,Y,U)),SOME([V,NAT]
                                                            DIFFPROP(Y,X,V)))
     Y * MORE        := SOME([U,NAT]DIFFPROP(X,Y,U))      ; PROP
     Y * LESS        := SOME([V,NAT]DIFFPROP(Y,X,V))      ; PROP
     Y * SATZ10B     := SATZ9B                            ; EC3(IS(X,Y),MORE(X,Y),
                                                            LESS(X,Y))
     Y * M           :=      ---                          ; MORE(X,Y)
     M * SATZ11      := M                                 ; LESS(Y,X)
     Y * MOREIS      := OR(MORE,IS(X,Y))                  ; PROP
     Y * LESSIS      := OR(LESS,IS(X,Y))                  ; PROP
     Y * M           :=      ---                          ; MOREIS(X,Y)
     M * SATZ13      := TH9'L-OR'(MORE,IS(X,Y),
                        LESS(Y,X),IS(Y,X),M,[Z,MORE]
                        SATZ11(Z),[Z,IS(X,Y)]
                        SYMIS(NAT,X,Y,Z))                 ; LESSIS(Y,X)
```

```
     Z * I           :=      ---                          ; IS(X,Y)
     I * M           :=      ---                          ; MORE(X,Z)
     M * ISMORE1     := ISP(NAT,[U,NAT]MORE(U,Z),X,Y,
                        M,I)                              ; MORE(Y,Z)
     I * M           :=      ---                          ; MOREIS(X,Z)
     M * ISMOREIS1   := ISP(NAT,[U,NAT]MOREIS(U,Z),X,
                        Y,M,I)                            ; MOREIS(Y,Z)
     I * M           :=      ---                          ; MOREIS(Z,X)
     M * ISMOREIS2   := ISP(NAT,[U,NAT]MOREIS(Z,U),X,
                        Y,M,I)                            ; MOREIS(Z,Y)
     Y * I           :=      ---                          ; IS(X,Y)
     I * MOREISI2    := ORI2(MORE(X,Y),IS(X,Y),I)         ; MOREIS(X,Y)
     Y * M           :=      ---                          ; MORE(X,Y)
     M * MOREISI1    := ORI1(MORE(X,Y),IS(X,Y),M)         ; MOREIS(X,Y)
     Z * U           :=      ---                          ; NAT
     U * I           :=      ---                          ; IS(X,Y)
     I * J           :=      ---                          ; IS(Z,U)
     J * M           :=      ---                          ; MOREIS(X,Z)
     M * ISMOREIS12  := ISMOREIS2(Z,U,Y,J,
                        ISMOREIS1(X,Y,Z,I,M))             ; MOREIS(Y,U)
     Y * M           :=      ---                          ; MORE(X,Y)
     M * SATZ10G     := TH3'L-OR'(LESS(X,Y),IS(X,Y),
                        EC3E23(IS(X,Y),MORE(X,Y),
                        LESS(X,Y),SATZ10B,M),
                        EC3E21(IS(X,Y),MORE(X,Y),
                        LESS(X,Y),SATZ10B,M))             ; NOT(LESSIS(X,Y))
     Y * SATZ18      := SOMEI(NAT,[U,NAT]
                        DIFFPROP(PL(X,Y),X,U),Y,
                        REFIS(NAT,PL(X,Y)))               ; MORE(PL(X,Y),X)
     Z * M           :=      ---                          ; MORE(X,Y)

+319

     M * U           :=      ---                          ; NAT
     U * DU          :=      ---                          ; DIFFPROP(U)
    DU * T1          := TRIS(NAT,X,PL(Y,U),PL(U,Y),DU,
                        COMPL(Y,U))                       ; IS(X,PL(U,Y))
    DU * T2          := TR3IS(NAT,PL(X,Z),PL(PL(U,Y),
                        Z),PL(U,PL(Y,Z)),PL(PL(Y,Z),U)
                        ,ISPL1(X,PL(U,Y),Z,T1),
                        ASSPL1(U,Y,Z),COMPL(U,PL(Y,Z))
                        )                                 ; IS(PL(X,Z),PL(PL(Y,Z),U))
    DU * T3          := SOMEI(NAT,[V,NAT]
                        DIFFPROP(PL(X,Z),PL(Y,Z),V),U,
                        T2)                               ; MORE(PL(X,Z),PL(Y,Z))

-319

     M * SATZ19A     := SOMEAPP(NAT,[U,NAT]DIFFPROP(U)
                        ,M,MORE(PL(X,Z),PL(Y,Z)),
                        [U,NAT][V,DIFFPROP(U)]
                        T3'-319'(U,V))                    ; MORE(PL(X,Z),PL(Y,Z))
     Z * M           :=      ---                          ; MOREIS(X,Y)

+*319

     M * N           :=      ---                          ; MORE(X,Y)
     N * T4          := MOREISI1(PL(X,Z),PL(Y,Z),
                        SATZ19A(N))                       ; MOREIS(PL(X,Z),PL(Y,Z))
     M * I           :=      ---                          ; IS(X,Y)
     I * T5          := MOREISI2(PL(X,Z),PL(Y,Z),
                        ISPL1(X,Y,Z,I))                   ; MOREIS(PL(X,Z),PL(Y,Z))

-319

     M * SATZ19L     := ORAPP(MORE(X,Y),IS(X,Y),
                        MOREIS(PL(X,Z),PL(Y,Z)),M,
                        [U,MORE(X,Y)]T4'-319'(U),[U,IS
                        (X,Y)]T5'-319'(U))                ; MOREIS(PL(X,Z),PL(Y,Z))
     M * SATZ19M     := ISMOREIS12(PL(X,Z),PL(Z,X),
                        PL(Y,Z),PL(Z,Y),COMPL(X,Z),
                        COMPL(Y,Z),SATZ19L)               ; MOREIS(PL(Z,X),PL(Z,Y))
```

```
+324

     X * N          :=      ---                          ; NIS(X,1)
     N * U          :=      ---                          ; NAT
     U * I          :=      ---                          ; IS(X,<U>SUC)
     I * T1         := TRIS(NAT,X,<U>SUC,PL(1,U),I,
                       SATZ4G(U))                        ; IS(X,PL(1,U))
     I * T2         := ISMORE1(PL(1,U),X,1,
                       SYMIS(NAT,X,PL(1,U),T1),
                       SATZ18(1,U))                     ; MORE(X,1)
     N * T3         := SOMEAPP(NAT,[U,NAT]IS(X,<U>
                       SUC),SATZ3(X,N),MORE(X,1),
                       [U,NAT][V,IS(X,<U>SUC)]T2(U,V)
                       )                                ; MORE(X,1)

-324

     X * SATZ24     := TH2"L-OR"(MORE(X,1),IS(X,1),
                       [U,NIS(X,1)]T3"-324"(U))         ; MOREIS(X,1)
     X * SATZ24A    := SATZ13(X,1,SATZ24)                ; LESSIS(1,X)
     Y * M          :=      ---                          ; MORE(Y,X)

+325

     M * U          :=      ---                          ; NAT
     U * DU         :=      ---                          ; DIFFPROP(Y,X,U)
    DU * T1         := SATZ19M(U,1,X,SATZ24(U))          ; MOREIS(PL(X,U),PL(X,1))
    DU * T2         := ISMOREIS1(PL(X,U),Y,PL(X,1),
                       SYMIS(NAT,Y,PL(X,U),DU),T1)      ; MOREIS(Y,PL(X,1))

-325

     M * SATZ25     := SOMEAPP(NAT,[U,NAT]DIFFPROP(Y,
                       X,U),M,MOREIS(Y,PL(X,1)),
                       [U,NAT][V,DIFFPROP(Y,X,U)]
                       T2"-325"(U,V))                   ; MOREIS(Y,PL(X,1))
     Y * L          :=      ---                          ; LESS(Y,X)
     L * SATZ25B    := SATZ13(X,PL(Y,1),SATZ25(Y,X,L)
                       )                                ; LESSIS(PL(Y,1),X)
       * P          :=      ---                          ; [X,NAT]PROP
     P * N          :=      ---                          ; NAT

+327

     N * M          :=      ---                          ; NAT
     M * LBPROP     := IMP(<M>P,LESSIS(N,M))             ; PROP

-327

     N * LB         := ALL([X,NAT]LBPROP"-327"(X))       ; PROP
     N * MIN        := AND(LB,<N>P)                      ; PROP
     P * S          :=      ---                          ; SOME(P)

+*327

     S * N          :=      ---                          ; NAT
     N * T1         := [X,<N>P]SATZ24A(N)                ; LBPROP(1,N)
     S * T2         := [X,NAT]T1(X)                      ; LB(1)
     S * L          :=      ---                          ; [X,NAT]LB(X)
     L * Y          :=      ---                          ; NAT
     Y * YP         :=      ---                          ; <Y>P
    YP * T3         := SATZ18(Y,1)                       ; MORE(PL(Y,1),Y)
    YP * T4         := SATZ10G(PL(Y,1),Y,T3)             ; NOT(LESSIS(PL(Y,1),Y))
    YP * T5         := TH4"L-IMP"(<Y>P,LESSIS(PL(Y,1)
                       ,Y),YP,T4)                       ; NOT(LBPROP(PL(Y,1),Y))
    YP * T6         := TH1"L-ALL"(NAT,[X,NAT]
                       LBPROP(PL(Y,1),X),Y,T5)          ; NOT(LB(PL(Y,1)))
    YP * T7         := MP(LB(PL(Y,1)),CON,<PL(Y,1)>L,
                       T6)                              ; CON
     L * T8         := SOMEAPP(NAT,P,S,CON,[X,NAT][Y,
                       <X>P]T7(X,Y))                    ; CON
```

```
    S * N          :=      ---                          ; NON(NAT,[X,NAT]AND(LB(X),
                                                          NOT(LB(PL(X,1)))))
    N * M          :=      ---                          ; NAT
    M * L          :=      ---                          ; LB(M)
    L * T9         := <M>N                              ; NOT(AND(LB(M),NOT(LB(PL(M,1))
                                                          )))
    L * T10        := ET(LB(PL(M,1)),
                      TH3'L-AND'(LB(M),
                      NOT(LB(PL(M,1))),T9,L))          ; LB(PL(M,1))
    L * T11        := ISP(NAT,[X,NAT]LB(X),PL(M,1),
                      <M>SUC,T10,SATZ4A(M))            ; LB(<M>SUC)
    N * T12        := [X,NAT]INDUCTION([Y,NAT]LB(Y),
                      T2,[Y,NAT][Z,LB(Y)]T11(Y,Z),X)   ; [X,NAT]LB(X)
    S * T13        := [X,NON(NAT,[X,NAT]AND(LB(X),
                      NOT(LB(PL(X,1)))))]T8(T12(X))    ; SOME([X,NAT]AND(LB(X),
                                                          NOT(LB(PL(X,1)))))
    S * M          :=      ---                          ; NAT
    M * A          :=      ---                          ; AND(LB(M),NOT(LB(PL(M,1))))
    A * T14        := ANDE1(LB(M),NOT(LB(PL(M,1)))),
                      A)                               ; LB(M)
    A * T15        := ANDE2(LB(M),NOT(LB(PL(M,1)))),
                      A)                               ; NOT(LB(PL(M,1)))
    A * NMP        :=      ---                          ; NOT(<M>P)
  NMP * N          :=      ---                          ; NAT
    N * NP         :=      ---                          ; <N>P
   NP * T16        := MP(<N>P,LESSIS(M,N),NP,<N>T14)    ; LESSIS(M,N)
   NP * T17        := TH3'L-IMP'(IS(M,N),<M>P,NMP,
                      [X,IS(M,N)]ISP(NAT,P,N,M,NP,
                      SYMIS(NAT,M,N,X)))               ; NOT(IS(M,N))
   NP * T18        := ORE1(LESS(M,N),IS(M,N),T16,
                      T17)                             ; LESS(M,N)
   NP * T19        := SATZ25B(N,M,T18)                  ; LESSIS(PL(M,1),N)
  NMP * T20        := [X,NAT][Y,<X>P]T19(X,Y)           ; LB(PL(M,1))
  NMP * T21        := MP(LB(PL(M,1)),CON,T20,T15)       ; CON
    A * T22        := ET(<M>P,[X,NOT(<M>P)]T21(X))      ; <M>P
    A * T23        := ANDI(LB(M),<M>P,T14,T22)          ; MIN(M)

-327

    S * SATZ27     := TH6'L-SOME'(NAT,[X,NAT]
                      AND(LB(X),NOT(LB(PL(X,1)))),
                      [X,NAT]MIN(X),T13'-327',
                      [X,NAT][Y,AND(LB(X),
                      NOT(LB(PL(X,1))))]T23'-327'(X,
                      Y))                              ; SOME([X,NAT]MIN(P,X))

-N

-LANDAU

-EQ

-ST

-E

-L
```

Appendix 5. Two shortcomings of the first verifying program

The verifying program which first checked our AUT-QE text was conceived at the time when the language theory of AUTOMATH was still in its infancy. Actually the first satisfactory definition of AUT-QE only appeared afterwards. The program can therefore be seen as a formalization of an informal concept of the language in the programmer's mind. This concept, though informal, was quite clear; in fact it was proved afterwards that the main procedure is adequate and terminates ([vD], [vD2]).

Besides being correct, the program had to be efficient: verifying a text should be actually feasible (and not only theoretically possible). This requirement led the programmer to economize on substitution, as by substitution expressions tend to become longer, and also because in substitution an expression has to be scanned and completely rebuilt. Even after the program had been operational for a year, simplifications by avoiding substitution shortened the process time considerably.

However, in two places economy went a bit too far. It is well known that α-reduction, i.e. renaming of bound variables (which is a special case of substitution) is sometimes necessary in order to avoid *clash of variables*. It has been assumed by the programmer that α-reduction is superfluous if all binding variables of input expressions get different codes (see [Z1]).

Unfortunately, as has been shown by v. Daalen, this is not the case. Clash of variables may still occur in the following two ways:

i) When it is tried to establish $[x,A]B \overset{D}{=} [y,C]D$ this is done by $A \overset{D}{=} C$ and $B \overset{D}{=} [\![y/x]\!]D$ (see [Z1], 8.4.1). This gives wrong results when x is free in D. It would be correct to try $A \overset{D}{=} C$ and $[\![x/z]\!]B \overset{D}{=} [\![y/z]\!]D$, where z is a fresh variable.

   The fact that clash of variables may actually occur in this way is shown by the following example. We consider the (correct) book:

* n :=	PN	E type
* x :=	—	E n
x * y :=	—	E [t,n]n
y * a :=	PN	E n
* b :=	PN	E [t,n]n

   Suppose it has to be established, relative to this book, whether

   <[y,[p,n]n][x,n]a(x,y)>[u,[q,[r,n]n][s,n]n]<<b>u>u

   $\overset{D}{=}$ [z,n]a(z,[v,n]a(z,b)) .

It is easily seen that both expressions are correct, and that the first expression reduces by β-reductions to

$$[x,n]a(x,[x,n]a(x,b))$$

where, as it should be, in the subexpression [x,n]a(x,b) the second x is bound. Hence the expressions are not definitionally equal. The program will not discover this, because it will proceed to check

$$[x,n]a(x,[x,n]a(x,b)) \overset{D}{\equiv} [x,n]a(x,[v,n]a(x,b))$$

and then $a \equiv a$ $x \overset{D}{\equiv} x$ and $[x,n]a(x,b) \overset{D}{\equiv} [x,n]a(x,b)$.

ii) The claim (in [Zl], 5.1) that by β-reductions on expressions with distinct binding variables eventually no clash of variables can arise is not justified, as we show by another example: Consider the following (correct) book:

* n :=	PN	E type
* x :=	—	E [t,n]n
x * y :=	—	E n
y * a :=	PN	E n
* b :=	PN	E [t,n][u,n]n

Now

```
<[z,[i,n][j,n]n][y,n][x,n]a(<x>z,y)>
[u,[k,[l,n][m,n]n][p,n][q,n]n]<<b>u>u
```

reduces by β-reductions to

```
[y,n][x,n]a(<x>[y,n][x,n]a(<x>b,y),y)
     (1)    (2)     (3)
```

If we reduce this further, the x indicated by (2), which is bound by the abstraction indicated by (1), will be bound by the abstraction indicated by (3), since the expression reduces (in the verifying program) to

```
[y,n][x,n]a([x,n]a(<x>b,x),y)
     (1)    (3)       (2)
```

while it should reduce to

```
[y,n][x,n]a([v,n]a(<v>b,x),y)
     (1)    (3)       (2)
```

(where v is a new variable).

Appendix 6. Example of a text in AUT-68

```
  * PROP :=                          PN                    E type
  * A    :=                          —                     E PROP
A * ⊢    :=                          PN                    E type

  * S    :=                          —                     E type
S * P    :=                          —                     E [x,S]PROP
P * ALL  :=                          PN                    E PROP
P * ∀    :=                          ALL                   E PROP
P * a    :=                          —                     E S
a * u    :=                          —                     E ⊢(∀(P))
u * ALLe :=                          PN                    E ⊢(<a>P)
u * ∀e   :=                          ALLe                  E ⊢(<a>P)
P * u    :=                          —                     E [x,S]⊢(<x>P)
u * ALLi :=                          PN                    E ⊢(∀(P))
u * ∀i   :=                          ALLi                  E ⊢(∀(P))

P * B    :=                          —                     E PROP
B * A→B  :=          ALL(⊢(A),[x,⊢(A)]B)                   E PROP
B * u    :=                          —                     E ⊢(A→B)
u * v    :=                          —                     E ⊢(A)
u * →e   :=          ALLe(⊢(A),[x,⊢(A)]B,v,u)              E ⊢(B)
B * u    :=                          —                     E [x,⊢(A)]⊢(B)
u * →i   :=          ALLi(⊢(A),[x,⊢(A)]B,u)                E ⊢(A→B)

  * ⊥    :=          ALL(PROP,[x,PROP]x)                   E PROP
A * u    :=                          —                     E ⊢(⊥)
u * ⊥e   :=          ALLe(PROP,[x,PROP]x,A,u)              E ⊢(A)

A * ¬    :=                          A→⊥                   E PROP
B * A∨B  := ALL(PROP,[x,PROP]((A→x)→((B→x)→x)))            E PROP
B * X    :=                          —                     E PROP
X * u    :=                          —                     E ⊢(A∨B)
u * v    :=                          —                     E [x,⊢(A)]⊢(X)
v * w    :=                          —                     E [x,⊢(B)]⊢(X)
w * ∨e   := →e(B→X,X,→e(A→X,(B→X)→X,
            ALLe(PROP,[x,PROP]((A→x)→((B→x)→x)),
            X,u),→i(A,X,v)),→i(B,X,w))                     E ⊢(X)
```

```
B * u      :=                          —              E ⊢(A)
u * vi1    := ALLi(PROP,[x,PROP]((A→x)→((B→x)→x)),
              [x,PROP]→i(A→x,(B→x)→x,
              [y,⊢(A→x)]→i(B→x,x,
              [z,⊢(B→x)]→e(A,x,y,u))))              E ⊢(A∨B)
B * u      :=                          —              E ⊢(B)
u * vi2    := ALLi(PROP,[x,PROP]((A→x)→((B→x)→x)),
              [x,PROP]→i(A→x,(B→x)→x,
              [y,⊢(A→x)]→i(B→x,x,
              [z,⊢(B→x)]→e(B,x,z,u))))              E ⊢(A∨B)

P * SOME   := ALL(PROP,[x,PROP](∀([y,S](<y>P→x))→x))  E PROP
P * ∃      :=                        SOME             E PROP
P * X      :=                         —               E PROP
X * u      :=                         —               E ⊢(∃(P))
u * v      :=                         —               E [x,S][y,⊢(<x>P)]⊢(X)
v * SOMEe  := →e(∀([y,S](<y>P→X)),X,
              ALLe(PROP,[x,PROP](∀([y,S](<y>P→x))
              →x),X,u),∀i([y,S](<y>P→X),
              [y,S]→i(<y>P,X,<y>v)))               E ⊢(X)
v * ∃e     :=                         SOMEe            E ⊢(X)
a * u      :=                         —               E ⊢(<a>P)
u * SOMEi  := ALLi(PROP,[x,PROP](∀([y,S](<y>P→x))→x),
              [x,PROP]→i(∀([y,S](<y>P→x)),x,
              [z,⊢(∀([y,S](<y>P→x)))]→e(<a>P,x,
              ∀e([y,S](<y>P→x),a,z),u)))            E ⊢(∃(P))
u * ∃i     :=                         SOMEi            E ⊢(∃(P))

S * a      :=                          —               E S
a * b      :=                          —               E S
b * IS     := ALL([x,S]PROP,[p,[x,S]PROP](<a>p↔<b>p)) E PROP
b * a=b    :=                           IS             E PROP
a * ISi    := ALLi([x,S]PROP,[p,[x,S]PROP](<a>p↔<a>p),
              [p,[x,S]PROP]→i(<a>p,<a>p,[y,⊢(<a>p)]y)) E⊢(a=a)
a * REFIS  :=                           ISi            E ⊢(a=a)
a * =i     :=                           ISi            E ⊢(a=a)
a * ref=   :=                           ISi            E ⊢(a=a)
```

```
P * a          :=                —                    E S
a * b          :=                —                    E S
b * u          :=                —                    E ⊢(a=b)
u * v          :=                —                    E ⊢(<a>P)
v * ISe        := →e(<a>P,<b>P,ALLe([x,S]PROP,
                  [p,[x,S]PROP](<a>p↣<b>p),P,u),v)    E ⊢(<b>P)
v * SUBST.PRED :=              ISe                    E ⊢(<b>P)
v * =e         :=              ISe                    E ⊢(<b>P)

S * a          :=                —                    E S
a * b          :=                —                    E S
b * u          :=                —                    E ⊢(a=b)
u * SYM.IS     := =e([x,S](x=a),a,b,u,=i(a))          E ⊢(b=a)
u * sym=       :=            SYM.IS                   E ⊢(b=a)
b * c          :=                —                    E S
e * u          :=                —                    E ⊢(a=b)
u * v          :=                —                    E ⊢(b=c)
v * TR.IS      := =e([x,S](a=x),b,c,v,u)              E ⊢(a=c)
v * tr=        :=            TR.IS                    E ⊢(a=c)

b * a≠b        :=           ¬(a=b)                    E PROP

S * T          :=                —                    E type
T * f          :=                —                    E [x,S]T
f * a          :=                —                    E S
a * b          :=                —                    E S
b * u          :=                —                    E ⊢(a=b)
u * SUBST.FN   := =e([x,S]IS(T,<a>f,<x>f),
                  a,b,u,ISi(T,<a>f))                  E ⊢(IS(T,<a>f,<b>f))

+N
  * nat        :=              PN                     E type
  * P          :=                —                    E [x,nat]PROP
P * ∀          :=      ALL(nat,P)                     E PROP
P * n          :=                —                    E nat
n * u          :=                —                    E ⊢(∀(P))
u * ∀e         :=      ALLe(nat,P,n,u)                E ⊢(<n>P)
```

P * u	:=	—	E [x,nat]⊢(<x>P)
u * ∀i	:=	ALLi(nat,P,u)	E ⊢(∀(P))
P * ∃	:=	SOME(nat,P)	E PROP
P * X	:=	—	E PROP
X * u	:=	—	E ⊢(∃(P))
u * v	:=	—	E [x,nat][y,⊢(<x>P)]⊢(X)
v * ∃e	:=	SOMEe(nat,P,X,u,v)	E ⊢(X)
n * u	:=	—	E ⊢(<n>P)
u * ∃i	:=	SOMEi(nat,P,n,u)	E ⊢(∃(P))
* n	:=	—	E nat
n * m	:=	—	E nat
m * n=m	:=	IS(nat,n,m)	E PROP
m * n≠m	:=	¬(n=m)	E PROP
n * ref=	:=	REF.IS(nat,n)	E ⊢(n=n)
m * u	:=	—	E ⊢(n=m)
u * sym=	:=	SYM.IS(nat,n,m,u)	E ⊢(m=n)
m * l	:=	—	E nat
l * u	:=	—	E ⊢(n=m)
u * v	:=	—	E ⊢(m=l)
v * tr=	:=	TR.IS(nat,n,m,l,u,v)	E ⊢(n=l)
P * n	:=	—	E nat
n * m	:=	—	E nat
m * u	:=	—	E ⊢(n=m)
u * v	:=	—	E ⊢(<n>P)
v* subst.pred	:=	SUBST.PRED(nat,P,n,m,u,v)	E ⊢(<m>P)
S * f	:=	—	E [x,nat]S
f * n	:=	—	E nat
n * m	:=	—	E nat
m * u	:=	—	E ⊢(n=m)
u * subst.fn	:=	SUBST.FN(nat,S,f,n,m,u)	E ⊢(IS(S,<n>f,<m>f))

```
  * 1          :=              PN              E nat
  * n          :=              —               E nat
n * n'         :=              PN              E nat

  * suc.fn     :=        [x,nat]x'             E [x,nat]nat

n * Axiom3     :=              PN              E ⊢(n'≠1)
n * m          :=              —               E nat
m * u          :=              —               E ⊢(n'=m')
u * Axiom4     :=              PN              E ⊢(n=m)

P * u          :=              —               E ⊢(<1>P)
u * v          :=              —               E [x,nat][y,⊢(<x>P)]⊢(<x'>P)
v * Axiom5     :=              PN              E ⊢(∀(P))

P * n          :=              —               E nat
n * u          :=              —               E ⊢(<1>P)
u * v          :=              —               E [x,nat][y,⊢(<x>P)]⊢(<x'>P)
v * induction := ∀e(P,n,Axiom5(P,u,v))         E ⊢(<n>P)

  * n          :=              —               E nat
n * m          :=              —               E nat
m * u          :=              —               E ⊢(n≠m)
u * Satz1      := →i(n'=m',⊥,[x,⊢(n'=m')]
                  →e(n=m,⊥,u,Axiom4(n,m,x))    E ⊢(n'≠m')

  * P2         := [x,nat](x'≠x)                E [x,nat]PROP
n * Satz2      := induction(P2,n,Axiom3(1),
                  [x,nat][y,⊢(<x>P2)]
                  Satz1(x',x,y))               E ⊢(n'≠n)

  * P3         := [x,nat]((x=1)∨∃([y,nat]
                  (x=y')))                     E [x,nat]PROP
  * l1         := ∨i1(1=1,∃([y,nat](1=y')),
                  ref=(1))                     E ⊢(<1>P3)
n * l2         := ∃i([y,nat](n'=y'),n,ref=(n')) E ⊢(∃([y,nat](n'=y'))
n * l3         := ∨i2(n'=1,∃([y,nat](n'=y')),
                  l2)                          E ⊢(<n'>P3)
```

```
n * ¬4    := induction(P3,n,¬1,[x,nat]
             [y,⊢(<x>P3]¬3(x))                    E ⊢(<n>P3)
n * u     :=              —                       E ⊢(n≠1)
u * Satz3 := ∨e(n=1,∃([y,nat](n=y')),
             ∃([y,nat](n=y')),¬4,[x,⊢(n=1)]⊥e
             (∃([y,nat](n=y')),→e(n=1,⊥,u,x)),
             [x,⊢(∃([y,nat](n=y')))]x)            E ⊢(∃([y,nat](n=y')))
```

Appendix 7. Excerpt for "Satz 1", "Satz 2" and "Satz 3".

```
LAYOUT FROM FILE : EXCERPTOUTPUT/SATZ1EN2EN3      JANUARY 25, 1977     10:52:35


+L

       * A          :=       ---                          # PROP
     A * B          :=       ---                          # PROP
     B * IMP        := [X,A]B                             # PROP
     B * C          :=       ---                          # PROP
     C * I          :=       ---                          # IMP(A,B)
     I * J          :=       ---                          # IMP(B,C)
     J * TRIMP      := [X,A]<<X>I>J                       # IMP(A,C)
       * CON        :=       PN                           # PROP
     A * NOT        := IMP(CON)                           # PROP
     A * WEL        := NOT(NOT(A))                        # PROP
     A * A1         :=       ---                          # A
    A1 * WELI       := [X,NOT(A)]<A1>X                    # WEL(A)
     A * W          :=       ---                          # WEL(A)
     W * ET         :=       PN                           # A
     A * C1         :=       ---                          # CON
    C1 * CONE       := ET([X,NOT(A)]C1)                   # A

+IMP

     B * N          :=       ---                          # NOT(A)
     N * TH2        := TRIMP(CON,B,N,[X,CON]CONE(B,X)
                       )                                  # IMP(A,B)
     B * N          :=       ---                          # NOT(B)
     N * I          :=       ---                          # IMP(A,B)
     I * TH3        := TRIMP(CON,I,N)                     # NOT(A)

-IMP

     B * OR         := IMP(NOT(A),B)                      # PROP
     B * A1         :=       ---                          # A
    A1 * ORI1       := TH2"-IMP"(NOT(A),B,WELI(A1))       # OR(A,B)
     B * B1         :=       ---                          # B
    B1 * ORI2       := [X,NOT(A)]B1                       # OR(A,B)
     B * O          :=       ---                          # OR(A,B)
     O * N          :=       ---                          # NOT(A)
     N * ORE2       := <N>O                               # B
       * SIGMA      :=       ---                          # TYPE
 SIGMA * P          :=       ---                          # [X,SIGMA]PROP
     P * ALL        := P                                  # PROP

+ALL

     P * S          :=       ---                          # SIGMA
     S * N          :=       ---                          # NOT(<S>P)
     N * TH1        := [X,ALL(SIGMA,P)]<<S>X>N           # NOT(ALL(SIGMA,P))

-ALL

     P * NON        := [X,SIGMA]NOT(<X>P)                 # [X,SIGMA]PROP
     P * SOME       := NOT(NON(P))                        # PROP
     P * S          :=       ---                          # SIGMA
     S * SP         :=       ---                          # <S>P
    SP * SOMEI      := TH1"-ALL"(NON(P),S,WELI(<S>P,
                       SP))                               # SOME(SIGMA,P)

+E

 SIGMA * S          :=       ---                          # SIGMA
     S * T          :=       ---                          # SIGMA
     T * IS         :=       PN                           # PROP
     S * REFIS      :=       PN                           # IS(S,S)

-E
```

```
+*E

+ST

 SIGMA * SET        :=      PN                          ; TYPE
 SIGMA * S          :=      ---                         ; SIGMA
     S * SO         :=      ---                         ; SET
    SO * ESTI       :=      PN                          ; PROP
     P * SETOF      :=      PN                          ; SET
     P * S          :=      ---                         ; SIGMA
     S * SP         :=      ---                         ; <S>P
    SP * ESTII      :=      PN                          ; ESTI(S,SETOF(P))
     S * E          :=      ---                         ; ESTI(S,SETOF(P))
     E * ESTIE      :=      PN                          ; <S>P

+EQ

+LANDAU

+N

       * NAT        :=      PN                          ; TYPE
       * X          :=      ---                         ; NAT
     X * Y          :=      ---                         ; NAT
     Y * IS         := IS'E'(NAT,X,Y)                   ; PROP
     Y * NIS        := NOT(IS(X,Y))                     ; PROP
     X * S          :=      ---                         ; SET(NAT)
     S * IN         := ESTI(NAT,X,S)                    ; PROP
       * P          :=      ---                         ; [X,NAT]PROP
     P * SOME       := SOME'L'(NAT,P)                   ; PROP
     P * ALL        := ALL'L'(NAT,P)                    ; PROP
       * 1          :=      PN                          ; NAT
       * SUC        :=      PN                          ; [X,NAT]NAT
       * AX3        :=      PN                          ; [X,NAT]NIS(<X>SUC,1)
       * AX4        :=      PN                          ; [X,NAT][Y,NAT][U,IS(<X>SUC,
                                                          <Y>SUC)]IS(X,Y)
       * S          :=      ---                         ; SET(NAT)
     S * COND1      := IN(1,S)                          ; PROP
     S * COND2      := ALL([X,NAT]IMP(IN(X,S),IN(<X>
                       SUC,S)))                         ; PROP
       * AX5        :=      PN                          ; [S,SET(NAT)][U,COND1(S)]
                                                          [V,COND2(S)][X,NAT]IN(X,S)
       * P          :=      ---                         ; [X,NAT]PROP
     P * 1P         :=      ---                         ; <1>P
    1P * XSP        :=      ---                         ; [X,NAT][Y,<X>P]<<X>SUC>P
   XSP * X          :=      ---                         ; NAT

+I1

     X * S          := SETOF(NAT,P)                     ; SET(NAT)
     X * T1         := ESTII(NAT,P,1,1P)                ; COND1(S)
     X * Y          :=      ---                         ; NAT
     Y * YES        :=      ---                         ; IN(Y,S)
   YES * T2         := ESTIE(NAT,P,Y,YES)               ; <Y>P
   YES * T3         := ESTII(NAT,P,<Y>SUC,<T2><Y>XSP)   ; IN(<Y>SUC,S)
     X * T4         := <X><[Y,NAT][U,IN(Y,S)]T3(Y,U)>
                       <T1><S>AX5                       ; IN(X,S)

-I1

     X * INDUCTION  := ESTIE(NAT,P,X,T4'-I1')           ; <X>P
       * X          :=      ---                         ; NAT
     X * Y          :=      ---                         ; NAT
     Y * N          :=      ---                         ; NIS(X,Y)

+21

     N * I          :=      ---                         ; IS(<X>SUC,<Y>SUC)
     I * T1         := <I><Y><X>AX4                     ; IS(X,Y)

-21

     N * SATZ1      := TH3'L-IMP'(IS(<X>SUC,<Y>SUC),
                       IS(X,Y),N,[U,IS(<X>SUC,<Y>SUC)
                       ]T1'-21'(U))                     ; NIS(<X>SUC,<Y>SUC)
```

```
+22

     X * PROP1      := NIS(<X>SUC,X)                    ; PROP
       * T1         := <1>AX3                           ; PROP1(1)
     X * P          :=      ---                         ; PROP1(X)
     P * T2         := SATZ1(<X>SUC,X,P)                ; PROP1(<X>SUC)

-22

     X * SATZ2      := INDUCTION([Y,NAT]PROP1"-22"(Y)
                       ,T1"-22",[Y,NAT][U,PROP1"-22"
                       (Y)]T2"-22"(Y,U),X)             ; NIS(<X>SUC,X)

+23

     X * PROP1      := OR(IS(X,1),SOME([U,NAT]IS(X,
                       <U>SUC)))                       ; PROP
       * T1         := ORI1(IS(1,1),SOME([U,NAT]IS(1,
                       <U>SUC)),REFIS(NAT,1))          ; PROP1(1)
     X * T2         := SOMEI(NAT,[U,NAT]IS(<X>SUC,<U>
                       SUC),X,REFIS(NAT,<X>SUC))       ; SOME([U,NAT]IS(<X>SUC,<U>SUC)
                                                         )
     X * T3         := ORI2(IS(<X>SUC,1),SOME([U,NAT]
                       IS(<X>SUC,<U>SUC)),T2)          ; PROP1(<X>SUC)
     X * T4         := INDUCTION([Y,NAT]PROP1(Y),T1,
                       [Y,NAT][U,PROP1(Y)]T3(Y),X)     ; PROP1(X)

-23

     X * N          :=      ---                         ; NIS(X,1)
     N * SATZ3      := ORE2(IS(X,1),SOME([U,NAT]IS(X,
                       <U>SUC)),T4"-23",N)             ; SOME([U,NAT]IS(X,<U>SUC))

-N

-LANDAU

-EQ

-ST

-E

-L
```

Appendix 8. Example of a text in AUT-68-SYNT

```
  * PROP      :=                    PN                  E type
  * A         :=                    —                   E PROP
A * ⊢         :=                    PN                  E type

  * z1        :=                    —                   E synt
z1 * ass.prop := lastelt(tail(⊢,cat(z1)))

So: if  u E ⊢(A)  then  ass.prop(u) ≝ A E PROP

  * S         :=                    —                   E type
S * P         :=                    —                   E [x,S]PROP
P * ALL       :=                    PN                  E PROP

z1 * ∀        :=            ALL(dom(z1),z1)

So: if  P E [x,S]PROP  then  ∀(P) ≝ ALL(S,P) E PROP

P * a         :=                    —                   E S
a * u         :=                    —                   E ⊢(∀(P))
u * Alle      :=                    PN                  E ⊢(<a>P)

z1 * z2       :=                    —                   E synt
z2 * ∀e       := ALLe(cat(z1),lastelt(tail(∀,ass.prop(z2))),
                 z1,z2)

So: if  a E S ,  u E ⊢(∀(P))  then  ∀e(a,u) E ⊢(<a>P)

P * u         :=                    —                   E [x,S]⊢(<x>P)
u * ALLi      :=                    PN                  E ⊢(∀(P))

z1 * ∀i       := ALLi(dom(z1),[x,dom(z1)]ass.prop(<x>z1),z1)

So: if  u E [x,S]⊢(<x>P)  then  ∀i(u) E ⊢(∀(P))

z1 * ∀2       := ∀([x,dom(z1)]∀(<x>z1))

So: if  P2 E [x,S][y,T(x)]PROP  then  ∀2(P2) ≝ ∀([x,S]∀([y,T(x)]
<y><x>P2)) E PROP
```

z2 * z3 := — E synt
z3 * ∀2e := ∀e(z2,∀e(z1,z3))

So: if a E S , b E T(a) , u E ⊢(∀2(P2)) then ∀2e(a,b,u) E ⊢(<b><a>P2)

z1 * ∀2i := ∀i([x,dom(z1)]∀i(<x>z1))

So: if u E [x,S][y,T(x)]⊢(<y><x>P2) then ∀2i(u) E ⊢(∀2(P2))

z1 * ∀3 := ∀([x,dom(z1)]∀2(<x>z1))

So: if P3 E [x,S][y,T(x)][z,U(x,y)]PROP then ∀3(P3) $\overset{D}{=}$ ∀([x,S]∀([y,T(x)]
∀([z,U(x,y)]<z><y><x>P3))) E PROP

z3 * z4 := — E synt
z4 * ∀3e := ∀2e(z2,z3,∀e(z1,z4))

So: if a E S , b E T(a) , c E U(a,b) , u E ⊢(∀3(P3)) then
∀3e(a,b,c,u) E ⊢(<c><b><a>P3)

z1 * ∀3i := ∀i([x,dom(z1)]∀2i(<x>z1))

So: if u E [x,S][y,T(x)][z,U(x,y)]⊢ (<z><y><x>P3) then ∀3i(u) E ⊢(∀3(P3))

A * B := — E PROP
B * A→B := ∀([x,⊢(A)]B) E PROP

z2 * →e := ∀e(z2,z1)
z2 * mod.pon := →e

So: if u E ⊢(A→B), v E ⊢(A) then →e(u,v) E ⊢(B), mod.pon(u,v) E ⊢(B).

* ⊥ := ∀([x,PROP]x) E PROP

A * ¬ := A→⊥ E PROP

B * A∨B := ∀([x,PROP]((A→x)→((B→x)→x))) E PROP

B * X := — E PROP
X * u := — E ⊢(A∨B)
u * v := — E [x,⊢(A)]⊢(X)
v * w := — E [x,⊢(B)]⊢(X)
w * ORe := ∀3e(X,∀i(v),∀i(w),u) E ⊢(X)
z3
z3 * ve := ORe(LFE(v,ass.prop(z1)),RFE(v,ass.prop(z1)),
lastelt(tail(⊢,val(cat(z2)))),z1,z2,z3)

So: if u E ⊢(A∨B), v E [x,⊢(A)]⊢(X), w E [x,⊢(B)]⊢(X) then ve(u,v,w) E⊢(X)

B * u := — E ⊢(A)
u * ORi1 := ∀3i([x,PROP][y,⊢(A→x)][z,⊢(B→x)]→e(y,u))E ⊢(A∨B)
B * u := — E ⊢(B)
u * ORi2 := ∀3i([x,PROP][y,⊢(A→x)][z,⊢(B→x)]→e(z,u)) E⊢(A∨B)

z2 * vi1 := ORi1(ass.prop(z2),z1,z2)
z2 * vi2 := ORi2(z1,ass.prop(z2),z2)

So: if B E PROP , u E ⊢(A) then vi1(B,u) E ⊢(A∨B)
if A E PROP , u E ⊢(B) then vi2(A,u) E ⊢(A∨B)

P * SOME := ∀([x,PROP](∀([y,S](<y>P→x))→x)) E PROP

z1* ∃ := SOME(dom(z1),z1)

So: if P E [x,S]PROP then ∃(P) $\stackrel{D}{=}$ SOME(S,P) E PROP

P * X := — E PROP
X * u := — E ⊢(∃(P))
u * v := — E [x,S][y,⊢(<x>P)]⊢(X)
v * SOMEe := ∀2e(X,∀2i(v),u) E ⊢(X)

z2 * ∃e := SOMEe(dom(z2),lastelt(tail(∃,ass.prop(z))),
lastelt(tail(⊢,val([x,dom(z2)]val(<x>cat(z2))))),z1,z2)

So: if u E (∃(P)), v E [x,S][y,⊢(<x>P)]⊢(X) then ∃e(u,v) E ⊢(X)

a * u := —— E ⊢(<a>P)
u * SOMEi := ∀2i([x,PROP][z,⊢(∀([y,S](<y>P→x)))]
∀2e(a,u,z)) E ⊢(∃(P))

z3 * ∃i := SOMEi(dom(z1),z1,z2,z3)

So: if P E [x,S]PROP, a E S, u E ⊢(<a>P) then ∃i(P,a,u) E ⊢(∃(P))

S * a := —— E S
a * b := —— E S
b * IS := ∀([p,[x,S]PROP](<a>p→<b>p)) E PROP

z2 * z1=z2 := IS(cat(z1),z1,z2)

So: if a E S, b E S then a=b $\overset{D}{=}$ IS(S,a,b) E PROP

z1 * left= := LFE(=,ass.prop(z1))
z1 * right= := RFE(=,ass.prop(z1))

So: if u E ⊢(a=b) then left=(u) $\overset{D}{=}$ a, right=(u) $\overset{D}{=}$ b

a * ISi := ∀2i([p,[x,S]PROP][y,⊢(<a>p)]y) E ⊢(a=a)

z1 * =i := ISi(cat(z1),z1)
z1 * ref= := =i

So: if a E S then =i(a) E ⊢(a=a), ref=(a) E ⊢(a=a)

P * a := —— E S
a * b := —— E S
b * u := —— E ⊢(a=b)
u * v := —— E ⊢(<a>P)
v * ISe := ∀2e(P,v,u) E ⊢(<b>P)

z3 * =e := ISe(dom(z1),z1,left=(z2), right=(z2),z2,z3)
z3 * subst.pred := =e

So: if P E [x,S]PROP, u E ⊢(a=b), v E ⊢(<a>P) then =e(P,u,v) E ⊢(<b>P), subst.pred(P,u,v) E ⊢(<b>P)

S * a	:=	—	E S
a * b	:=	—	E S
b * u	:=	—	E ⊢(a=b)
u * SYM.IS	:= =e([x,S](x=a),u,=i(a))		E ⊢(b=a)

z1 * sym= := SYM.IS(cat(left=(z1)),left=(z1),right=(z1),z1)

So: if u E ⊢(a=b) then sym=(u) E ⊢(b=a)

b * c	:=	—	E S
c * u	:=	—	E ⊢(a=b)
u * v	:=	—	E ⊢(b=c)
v * TR.IS	:= =e([x,S](a=x),v,u)		E ⊢(a=c)

z1 * tr= := TR.IS(cat(left=(z1)),left=(z1),right=(z1),right=(z2),z1,z2)

So: if u E ⊢(a=b), v E ⊢(b=c) then tr=(u,v) E ⊢(a=c)

z2 * z1≠z2 := ¬(z1=z2)

So: if a E S, b E S then a≠b $\stackrel{D}{=}$ ¬(a=b) E PROP

S * T	:=	—	E type
T * f	:=	—	E [x,S]T
f * a	:=	—	E S
a * b	:=	—	E S
b * u	:=	—	E ⊢(a=b)
u * SUBST.FN	:= =e([x,S](<a>f = <x>f),u,=i(<a>f))		E ⊢(<a>f=<b>f)

z2 * subst.fn := SUBST.FN(dom(z1),val(cat(z1)),z1,left=(z2),right=(z2),z2)

So: if f E [x,S]T, u E ⊢(a=b) then subst.fn(f,u) E ⊢(<a>f = <b>f)

```
  * nat       :=            PN              E type
  * 1         :=            PN              E nat
  * n         :=            —               E nat
n * n'        :=            PN              E nat

  * suc.fn    :=       [x,nat]x'            E [x,nat]nat

n * AXIOM3    :=            PN              E ⊢(n'≠1)
n * m         :=            —               E nat
m * u         :=            —               E ⊢(n'=m')
u * AXIOM4    :=            PN              E ⊢(n=m)

  * P         :=            —               E [x,nat]PROP
P * u         :=            —               E ⊢(<1>P)
u * v         :=            —               E [x,nat][y,⊢(<x>P)]⊢(<x'>P)
v * AXIOM5    :=            PN              E ⊢(∀(P))
n * Axiom3    :=          AXIOM3            E ⊢(n'≠1)

z1 * Axiom4   := AXIOM4(FE(',right=(z1)),FE(',left=(z1)),z1)

So: if  u E ⊢(n'=m')  then  axiom4(u) E ⊢(n=m)

v * Axiom5    :=          AXIOM5            E ⊢(∀(P))

P * n         :=            —               E nat
n * u         :=            —               E ⊢(<1>P)
u * v         :=            —               E [x,nat][y,⊢(<x>P)]⊢(<x'>P)
v * induction := ∀e(n,Axiom5(P,u,v))        E ⊢(<n>P)

  * n         :=            —               E nat
n * m         :=            —               E nat
m * u         :=            —               E ⊢(n≠m)
u * SATZ1     := ∀i([x,⊢(n'=m')]
                  →e(u,Axiom4(x)))          E ⊢(n'≠m')

z1 * Satz1    := SATZ1(LFE(≠,ass.prop(z1)),RFE(≠,ass.prop(z1)),z1)

So: if  u E ⊢(n≠m)  then  Satz1(u) E ⊢(n'≠m')
```

```
  * P2     := [x,nat](x'≠x)                          E [x,nat]PROP
n * Satz2  := induction(P2,n,Axiom3(1),[x,nat]
              [y,⊢(<x>P2)]Satz1(y))                  E ⊢(n'≠n)

  * P3     := [x,nat]((x=1)∨∃([y,nat](x=y')))        E [x,nat]PROP
  * l1     := ∨i1(∃([y,nat](1=y')),ref=(1))          E ⊢(<1>P3)
n * l2     := ∃i([y,nat](n'=y'),n,ref=(n'))          E ⊢(∃([y,nat](n'=y')))
n * l3     := ∨i2(n'=1,l2)                           E ⊢(<n'>P3)
n * l4     := induction(P3,n,l3,[x,nat]
              [y,⊢(<x>P3)]l3(x))                     E ⊢(<n>P3)
n * u      :=                  —                     E ⊢(n≠1)
u * SATZ3  := ∨e(l4,[x,⊢(n=1)]∀2e(x,∃([y,nat]
              (n=y')),u),[x,⊢(∃([y,nat](n=y')))]x)   E ⊢(∃([y,nat](n=y')))

z1 * Satz3 := SATZ3(LFE(≠,ass.prop(z1)),z1)

So: if  u E ⊢(n≠1)  then  Satz3(u) E ⊢(∃([y,nat](n=y')))
```

Appendix 9. AUT-SYNT

In 4.1.0 we have indicated that for andi the parameters u and v are essential, while a and b are redundant parameters. If A, B, p and q can be correctly substituted for a , b , u and v , then A and B can be calculated (up to definitional equality) from p and q, because A is definitionally equal to CAT(p) and B to CAT(q).

Here we introduce an extension of AUTOMATH languages, called AUT-SYNT, in which it is possible to suppress redundant parameters. In this language, CAT is incorporated as a *predefined function*. For any 2- or 3-expression E, CAT(E) is the mechanically calculated type of E. It follows that andi(CAT(p),CAT(q),p,q) equals andi(A,B,p,q). The extended language moreover contains *variables for expressions*. A basic symbol synt (which has no degree) is added to the language. Variables of type synt (or synt variables) are to be interpreted as syntactic variables for expressions. There are no typing restrictions on substitution for such a variable.

Following the AUT-QE text in 4.1.1 we can write in AUT-SYNT:

```
    * z1   :=          —          E synt
z1 * z2   :=          —          E synt
z2 * ANDI := andi(CAT(z1),CAT(z2),z1,z2)
```

Now, if A E prop, B E prop, p E A, q E B then ANDI(p,q) = andi(CAT(p), CAT(q),p,q) = andi(A,B,p,q) E and(a,b).

Besides CAT we have other predefined functions in AUT-SYNT. They are defined for certain classes of expressions (just as CAT is defined for 2-expressions and 3-expressions). We list these functions here with their semantics. In the description of the semantics we will frequently use the clause: "if E reduces to ...". We will say e.g. "if E reduces to [x,A]B...". This is intended to mean: "if [x,A]B is the first abstraction expression in the reduction sequence, obtained by reducing E according to the strategy of the verifying program". Similar meanings are intended in other cases. Everywhere in the description E and $E_1,E_2,\ldots,E_n$ will denote correct AUT-expressions.

predefined function	semantics
CAT	CAT(E) is the "mechanical type" of the 2- or 3-expression E
DOM	If E reduces to [x,A]B or CAT(E) reduces to [x,A]B or CAT(CAT(E)) = [x,A]B then DOM(E) = A.

VAL	If E reduces to [x,A]B and B does not contain x then VAL(E) = B.
ARG	If E reduces to <A>B then ARG(E) = A.
FUN	If E reduces to <A>B then FUN(E) = B.
TAIL	If E reduces to $c(A_1,\ldots,A_n)$ then TAIL(c,E) is the string of expressions $A_1,\ldots,A_n$.
LASTELT	If $E_1,\ldots,E_n$ is a nonempty string of expressions then $\text{LASTELT}(E_1,\ldots,E_n) = E_n$.
PREPART	If $E_1,\ldots,E_n$ is a nonempty string of expressions then $\text{PREPART}(E_1,\ldots,E_n)$ is the string of expressions $E_1,\ldots,E_{n-1}$.

Remarks:

1) Expressions containing synt variables do not have a type. Lines having such an expression as their middle part do not have a category part.
2) EB-lines which have synt variables in their context can only have synt as their category part. In other words: on a synt context only synt variables may be introduced.
3) The identifiers CAT, DOM, VAL, ARG, FUN, TAIL, PREPART and LASTELT, and the identifiers defined in terms of these should not be treated as ordinary identifiers. In particular the monotonicity of definitional equality (in this case: if A = B then c(A) = c(B) where c is one of these special identifiers) does not generally apply here. E.g. if f = [x,nat]1 then <1>f $\overset{D}{=}$ <<1>suc>f, while ARG(<1>f) = 1 $\overset{D}{\neq}$ <1>suc = ARG(<<1>suc>f). Similar examples can be found for FUN and TAIL.
4) For languages admitting infix expressions there are functions LFE (for left fixed expression) and RFE (for right fixed expression) with semantics:
   If E reduces to A c B then LFE(c,E) = A and RFE(c,E) = B.

Examples:

1) The first elimination rule for conjunction can be represented in AUT-QE by adding, on the context a E prop ; b E prop ; introduced in 4.1.0:

```
b * u     :=        —          E and(a,b)
u * andel :=        ....       E a
```

Then a and b are redundant parameters, for andel and u is an essential parameter. In fact, if p is a substitution instance for u , then the type of p can be expected to reduce to and(A,B) for some A and B, and these A and B should the be substituted for a and b .
Therefore, keeping the context z1 E synt introduced above, we can add the AUT-SYNT line

```
z1 * ANDE1 := andel(LASTELT(PREPART(TAIL(and,CAT(z1)))),
                    LASTELT(TAIL(and(CAT(z1))),z1)
```

Then p E and(A,B) implies ANDE1(p) E A.
We can now indicate a complication which must be kept in mind when using AUT-SYNT, and which is connected with remark 3 above. Suppose and has been defined by and := not(imp(a,not(b))) . We may have p, A and B such that CAT(p) ≡ not(imp(A,not(B))), and then we have andel(A,B,p) E A, but ANDE1(p) will be incorrect, since CAT(p) does not *reduce* to and(A,B). Even worse complications may occur when using ARG and FUN.

2) In [vD, 3.6] book-equality is introduced. In AUT-SYNT we could add to this text, on the context z1 E synt ; z2 E synt ;

```
z2 * is    := IS(CAT(z1),z1,z2)
z1 * refis := REFIS(CAT(z1),z1)
z1   symis := SYMIS(CAT(LASTELT(TAIL(is,z1))),
                    LASTELT(PREPART(TAIL(is,z1))),
                    LASTELT(TAIL(is,z1)),z1)
```

Then for any type S , if x E S and y E S , equality of x and y could be expressed by is(x,y) instead of IS(S,x,y) . Moreover, if x E S we have refis(x) E is(x,x) and if p E is(x,y) we have symis(p) E is(y,x).

3) A text in AUT-68-SYNT, in which the first three theorems of Landau's book are proved, appears in appendix 8.

References

[dB] N.G. de Bruijn, AUTOMATH, a language for mathematics. Notes (prepared by B. Fawcett) of a series of lectures in the Séminaire de Mathématiques Supérieures.
Université de Montréal, 1971.

[dB2] N.G. de Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem.
Indag. Math., 34, 5, 1972.

[vD] 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.
Appendix 1 in this thesis.

[vD2] D.T. van Daalen, The language theory of AUTOMATH.
Thesis, Eindhoven University of Technology, to appear 1979.

[J] L.S. Jutting, A translation of Landau's "Grundlagen" in AUTOMATH.
Eindhoven University of Technology, Dept. of Math., 1976.

[L] E. Landau, Grundlagen der Analysis.
3rd ed., Chelsea Publ. Comp., New York, 1960.

[dV] R. de Vrijer, Big Trees in a λ-calculus with λ-expressions as types.
λ-calculus and Computer Science, ed. C. Böhm.
Springer, Berlin-Heidelberg - New York, 1975.

[Z1] I. Zandleven, A verifying program for AUTOMATH.
Proceedings of the Symposium on APL. ed. P. Braffort. Paris 1974.

[Z] J. Zucker, Formalization of classical mathematics in AUTOMATH.
To appear in: Actes du colloque international de logique, Clermont-Ferrand, July 1975, ed. M. Guillaume.
Preprint: Eindhoven University of Technology, Dept. of Math.

OTHER TITLES IN THE SERIES MATHEMATICAL CENTRE TRACTS

A leaflet containing an order-form and abstracts of all publications mentioned below is available at the Mathematisch Centrum, Tweede Boerhaavestraat 49, Amsterdam-1005, The Netherlands. Orders should be sent to the same address.

MCT 1 T. VAN DER WALT, *Fixed and almost fixed points*, 1963. ISBN 90 6196 002 9.

MCT 2 A.R. BLOEMENA, *Sampling from a graph*, 1964. ISBN 90 6196 003 7.

MCT 3 G. DE LEVE, *Generalized Markovian decision processes, part I: Model and method*, 1964. ISBN 90 6196 004 5.

MCT 4 G. DE LEVE, *Generalized Markovian decision processes, part II: Probabilistic background*, 1964. ISBN 90 6196 005 3.

MCT 5 G. DE LEVE, H.C. TIJMS & P.J. WEEDA, *Generalized Markovian decision processes, Applications*, 1970. ISBN 90 6196 051 7.

MCT 6 M.A. MAURICE, *Compact ordered spaces*, 1964. ISBN 90 6196 006 1.

MCT 7 W.R. VAN ZWET, *Convex transformations of random variables*, 1964. ISBN 90 6196 007 X.

MCT 8 J.A. ZONNEVELD, *Automatic numerical integration*, 1964. ISBN 90 6196 008 8.

MCT 9 P.C. BAAYEN, *Universal morphisms*, 1964. ISBN 90 6196 009 6.

MCT 10 E.M. DE JAGER, *Applications of distributions in mathematical physics*, 1964. ISBN 90 6196 010 X.

MCT 11 A.B. PAALMAN-DE MIRANDA, *Topological semigroups*, 1964. ISBN 90 6196 011 8.

MCT 12 J.A.TH.M. VAN BERCKEL, H. BRANDT CORSTIUS, R.J. MOKKEN & A. VAN WIJNGAARDEN, *Formal properties of newspaper Dutch*, 1965. ISBN 90 6196 013 4.

MCT 13 H.A. LAUWERIER, *Asymptotic expansions*, 1966, out of print; replaced by MCT 54 and 67.

MCT 14 H.A. LAUWERIER, *Calculus of variations in mathematical physics*, 1966. ISBN 90 6196 020 7.

MCT 15 R. DOORNBOS, *Slippage tests*, 1966. ISBN 90 6196 021 5.

MCT 16 J.W. DE BAKKER, *Formal definition of programming languages with an application to the definition of ALGOL 60*, 1967. ISBN 90 6196 022 3.

MCT 17 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 1*, 1968. ISBN 90 6196 025 8.

MCT 18 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 2*, 1968. ISBN 90 6196 038 X.

MCT 19 J. VAN DER SLOT, *Some properties related to compactness*, 1968. ISBN 90 6196 026 6.

MCT 20 P.J. VAN DER HOUWEN, *Finite difference methods for solving partial differential equations*, 1968. ISBN 90 6196 027 4.

MCT 21 E. WATTEL, *The compactness operator in set theory and topology*, 1968. ISBN 90 6196 028 2.

MCT 22 T.J. DEKKER, *ALGOL 60 procedures in numerical algebra, part 1*, 1968. ISBN 90 6196 029 0.

MCT 23 T.J. DEKKER & W. HOFFMANN, *ALGOL 60 procedures in numerical algebra, part 2*, 1968. ISBN 90 6196 030 4.

MCT 24 J.W. DE BAKKER, *Recursive procedures*, 1971. ISBN 90 6196 060 6.

MCT 25 E.R. PAERL, *Representations of the Lorentz group and projective geometry*, 1969. ISBN 90 6196 039 8.

MCT 26 EUROPEAN MEETING 1968, *Selected statistical papers, part I*, 1968. ISBN 90 6196 031 2.

MCT 27 EUROPEAN MEETING 1968, *Selected statistical papers, part II*, 1969. ISBN 90 6196 040 1.

MCT 28 J. OOSTERHOFF, *Combination of one-sided statistical tests*, 1969. ISBN 90 6196 041 X.

MCT 29 J. VERHOEFF, *Error detecting decimal codes*, 1969. ISBN 90 6196 042 8.

MCT 30 H. BRANDT CORSTIUS, *Excercises in computational linguistics*, 1970. ISBN 90 6196 052 5.

MCT 31 W. MOLENAAR, *Approximations to the Poisson, binomial and hypergeometric distribution functions*, 1970. ISBN 90 6196 053 3.

MCT 32 L. DE HAAN, *On regular variation and its application to the weak convergence of sample extremes*, 1970. ISBN 90 6196 054 1.

MCT 33 F.W. STEUTEL, *Preservation of infinite divisibility under mixing and related topics*, 1970. ISBN 90 6196 061 4.

MCT 34 I. JUHÁSZ, A. VERBEEK & N.S. KROONENBERG, *Cardinal functions in topology*, 1971. ISBN 90 6196 062 2.

MCT 35 M.H. VAN EMDEN, *An analysis of complexity*, 1971. ISBN 90 6196 063 0.

MCT 36 J. GRASMAN, *On the birth of boundary layers*, 1971. ISBN 90 6196 064 9.

MCT 37 J.W. DE BAKKER, G.A. BLAAUW, A.J.W. DUIJVESTIJN, E.W. DIJKSTRA, P.J. VAN DER HOUWEN, G.A.M. KAMSTEEG-KEMPER, F.E.J. KRUSEMAN ARETZ, W.L. VAN DER POEL, J.P. SCHAAP-KRUSEMAN, M.V. WILKES & G. ZOUTENDIJK, *MC-25 Informatica Symposium*, 1971. ISBN 90 6196 065 7.

MCT 38 W.A. VERLOREN VAN THEMAAT, *Automatic analysis of Dutch compound words*, 1971. ISBN 90 6196 073 8.

MCT 39 H. BAVINCK, *Jacobi series and approximation*, 1972. ISBN 90 6196 074 6.

MCT 40 H.C. TIJMS, *Analysis of (s,S) inventory models*, 1972. ISBN 90 6196 075 4.

MCT 41 A. VERBEEK, *Superextensions of topological spaces*, 1972. ISBN 90 6196 076 2.

MCT 42 W. VERVAAT, *Success epochs in Bernoulli trials (with applications in number theory)*, 1972. ISBN 90 6196 077 0.

MCT 43 F.H. RUYMGAART, *Asymptotic theory of rank tests for independence*, 1973. ISBN 90 6196 081 9.

MCT 44 H. BART, *Meromorphic operator valued functions*, 1973. ISBN 90 6196 082 7.

MCT 45 A.A. BALKEMA, *Monotone transformations and limit laws*, 1973. ISBN 90 6196 083 5.

MCT 46 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 1: The language*, 1973. ISBN 90 6196 084 3.

MCT 47 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 2: The compiler*, 1973. ISBN 90 6196 085 1.

MCT 48 F.E.J. KRUSEMAN ARETZ, P.J.W. TEN HAGEN & H.L. OUDSHOORN, *An ALGOL 60 compiler in ALGOL 60, Text of the MC-compiler for the EL-X8*, 1973. ISBN 90 6196 086 x.

MCT 49 H. KOK, *Connected orderable spaces*, 1974. ISBN 90 6196 088 6.

MCT 50 A. VAN WIJNGAARDEN, B.J. MAILLOUX, J.E.L. PECK, C.H.A. KOSTER, M. SINTZOFF, C.H. LINDSEY, L.G.L.T. MEERTENS & R.G. FISKER (Eds), *Revised report on the algorithmic language ALGOL 68*, 1976. ISBN 90 6196 089 4.

MCT 51 A. HORDIJK, *Dynamic programming and Markov potential theory*, 1974. ISBN 90 6196 095 9.

MCT 52 P.C. BAAYEN (ed.), *Topological structures*, 1974. ISBN 90 6196 096 7.

MCT 53 M.J. FABER, *Metrizability in generalized ordered spaces*, 1974. ISBN 90 6196 097 5.

MCT 54 H.A. LAUWERIER, *Asymptotic analysis, part 1*, 1974. ISBN 90 6196 098 3.

MCT 55 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 1: Theory of designs, finite geometry and coding theory*, 1974. ISBN 90 6196 099 1.

MCT 56 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 2: graph theory, foundations, partitions and combinatorial geometry*, 1974. ISBN 90 6196 100 9.

MCT 57 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 3: Combinatorial group theory*, 1974. ISBN 90 6196 101 7.

MCT 58 W. ALBERS, *Asymptotic expansions and the deficiency concept in statistics*, 1975. ISBN 90 6196 102 5.

MCT 59 J.L. MIJNHEER, *Sample path properties of stable processes*, 1975. ISBN 90 6196 107 6.

MCT 60 F. GÖBEL, *Queueing models involving buffers*, 1975. ISBN 90 6196 108 4.

* MCT 61 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 1*. ISBN 90 6196 109 2.

* MCT 62 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 2*. ISBN 90 6196 110 6.

MCT 63 J.W. DE BAKKER (ed.), *Foundations of computer science*, 1975. ISBN 90 6196 111 4.

MCT 64 W.J. DE SCHIPPER, *Symmetric closed categories*, 1975. ISBN 90 6196 112 2.

MCT 65 J. DE VRIES, *Topological transformation groups 1 A categorical approach*, 1975. ISBN 90 6196 113 0.

MCT 66 H.G.J. PIJLS, *Locally convex algebras in spectral theory and eigenfunction expansions*, 1976. ISBN 90 6196 114 9.

* MCT 67 H.A. LAUWERIER, *Asymptotic analysis, part 2.*
ISBN 90 6196 119 X.

MCT 68 P.P.N. DE GROEN, *Singularly perturbed differential operators of second order*, 1976. ISBN 90 6196 120 3.

MCT 69 J.K. LENSTRA, *Sequencing by enumerative methods*, 1977.
ISBN 90 6196 125 4.

MCT 70 W.P. DE ROEVER JR., *Recursive program schemes: semantics and proof theory*, 1976. ISBN 90 6196 127 0.

MCT 71 J.A.E.E. VAN NUNEN, *Contracting Markov decision processes*, 1976.
ISBN 90 6196 129 7.

MCT 72 J.K.M. JANSEN, *Simple periodic and nonperiodic Lamé functions and their applications in the theory of conical waveguides*, 1977.
ISBN 90 6196 130 0.

MCT 73 D.M.R. LEIVANT, *Absoluteness of intuitionistic logic*, 1979.
ISBN 90 6196 122 x.

MCT 74 H.J.J. TE RIELE, *A theoretical and computational study of generalized aliquot sequences*, 1976. ISBN 90 6196 131 9.

MCT 75 A.E. BROUWER, *Treelike spaces and related connected topological spaces*, 1977. ISBN 90 6196 132 7.

MCT 76 M. REM, *Associons and the closure statement*, 1976. ISBN 90 6196 135 1.

MCT 77 W.C.M. KALLENBERG, *Asymptotic optimality of likelihood ratio tests in exponential families*, 1977 ISBN 90 6196 134 3.

MCT 78 E. DE JONGE, A.C.M. VAN ROOIJ, *Introduction to Riesz spaces*, 1977.
ISBN 90 6196 133 5.

MCT 79 M.C.A. VAN ZUIJLEN, *Empirical distributions and rankstatistics*, 1977.
ISBN 90 6196 145 9.

MCT 80 P.W. HEMKER, *A numerical study of stiff two-point boundary problems*, 1977. ISBN 90 6196 146 7.

MCT 81 K.R. APT & J.W. DE BAKKER (eds), *Foundations of computer science II*, part I, 1976. ISBN 90 6196 140 8.

MCT 82 K.R. APT & J.W. DE BAKKER (eds), *Foundations of computer science II*, part II, 1976. ISBN 90 6196 141 6.

MCT 83 L.S. VAN BENTEM JUTTING, *Checking Landau's "Grundlagen" in the AUTOMATH system*, ISBN 90 6196 147 5.

MCT 84 H.L.L. BUSARD, *The translation of the elements of Euclid from the Arabic into Latin by Hermann of Carinthia (?) books vii-xii*, 1977.
ISBN 90 6196 148 3.

MCT 85 J. VAN MILL, *Supercompactness and Wallman spaces*, 1977.
ISBN 90 6196 151 3.

MCT 86 S.G. VAN DER MEULEN & M. VELDHORST, *Torrix I*, 1978.
ISBN 90 6196 152 1.

* MCT 87 S.G. VAN DER MEULEN & M. VELDHORST, *Torrix II*,
ISBN 90 6196 153 x.

MCT 88 A. SCHRIJVER, *Matroids and linking systems*, 1977.
ISBN 90 6196 154 8.

MCT 89 J.W. DE ROEVER, *Complex Fourier transformation and analytic functionals with unbounded carriers*, 1978. ISBN 90 6196 155 6.

* MCT 90 L.P.J. GROENEWEGEN, *Characterization of optimal strategies in dynamic games*, . ISBN 90 6196 156 4.

* MCT 91 J.M. GEYSEL, *Transcendence in fields of positive characteristic*, . ISBN 90 6196 157 2.

* MCT 92 P.J. WEEDA, *Finite generalized Markov programming*, . ISBN 90 6196 158 0.

MCT 93 H.C. TIJMS (ed.) & J. WESSELS (ed.), *Markov decision theory*, 1977. ISBN 90 6196 160 2.

MCT 94 A. BIJLSMA, *Simultaneous approximations in transcendental number theory*, 1978 . ISBN 90 6196 162 9.

MCT 95 K.M. VAN HEE, *Bayesian control of Markov chains*, 1978. ISBN 90 6196 163 7.

* MCT 96 P.M.B. VITÁNYI, *Lindenmayer systems: structure, languages, and growth functions*, . ISBN 90 6196 164 5.

* MCT 97 A. FEDERGRUEN, *Markovian control problems; functional equations and algorithms*, . ISBN 90 6196 165 3.

MCT 98 R. GEEL, *Singular perturbations of hyperbolic type*, 1978. ISBN 90 6196 166 1

MCT 99 J.K. LENSTRA, A.H.G. RINNOOY KAN & P. VAN EMDE BOAS, *Interfaces between computer science and operations research*, 1978. ISBN 90 6196 170 X.

MCT 100 P.C. BAAYEN, D. VAN DULST & J. OOSTERHOFF (Eds), *Proceedings bicentennial congress of the Wiskundig Genootschap, part 1*, 1979. ISBN 90 6196 168 8.

MCT 101 P.C. BAAYEN, D. VAN DULST & J. OOSTERHOFF (Eds), *Proceedings bicentennial congress of the Wiskundig Genootschap, part 2*, 1979. ISBN 90 9196 169 6.

MCT 102 D. VAN DULST, *Reflexive and superreflexive Banach spaces*, 1978. ISBN 90 6196 171 8.

MCT 103 K. VAN HARN, *Classifying infinitely divisible distributions by functional equations*, 1978 . ISBN 90 6196 172 6.

MCT 104 J.M. VAN WOUWE, *Go-spaces and generalizations of metrizability*, 1979. ISBN 90 6196 173 4.

* MCT 105 R. HELMERS, *Edgeworth expansions for linear combinations of order statistics*, . ISBN 90 6196 174 2.

MCT 106 A. SCHRIJVER (Ed.), *Packing and covering in combinatories*, 1979. ISBN 90 6196 180 7.

MCT 107 C. DEN HEIJER, *The numerical solution of nonlinear operator equations by imbedding methods*, 1979. ISBN 90 6196 175 0.

* MCT 108 J.W. DE BAKKER & J. VAN LEEUWEN (Eds), *Foundations of computer science III*, part I, . ISBN 90 6196 176 9.

* MCT 109 J.W. DE BAKKER & J. VAN LEEUWEN (Eds), *Foundations of computer science III*, part II . ISBN 90 6196 177 7.

MCT 110 J.C. VAN VLIET, *ALGOL 68 transput*, part I,1979 . ISBN 90 6196 178 5.

MCT 111 J.C. VAN VLIET, *ALGOL 68 transput*, part II: *An implementation model*, 1979. ISBN 90 6196 179 3.

AN ASTERISK BEFORE THE NUMBER MEANS "TO APPEAR"