597921

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics

Memorandum 1976-14.

Issued December 1976.

Modifications of the 1968 version of AUTOMATH

by

N.G. de Bruijn.

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

Modifications of the 1968 version of AUTOMATH.

by N.G. de Bruijn.

1. Introduction. The 1968 version of AUTOMATH (AUT-68 for short) has been explained in several reports.

We quote [1],[2] for informal papers, mainly concentrating on the use of the language, to [5] for a more formal definition, and to [4] for its place within a certain family of languages.

As to that family it should be mentioned that one family member, AUT-QE, made itself quite popular, since it provides a first-order theory which nevertheless can deal adequately with sets and predicates.

The first use that was made of AUT-68 could treat propositions by means of a type called "bool", the members of which are to be interpreted as propositions. This puts us in a position to quantify over predicates, and to enjoy the power of a higher order theory without having it in the language itself. The strength of such a theory is that we are able to define equality and to prove some of its basic properties instead of having to take them as primitive notion and axioms, respectively.

The preference developed later for AUT-QE (which L.S.Jutting selected for the translation of Landau's "Grundlagen") was, however, not so much based on comtempt for a theory that seemed to be stronger than necessary for the purpose of talking ordinary mathematics. The real reason was that the use of "bool" in AUT-68 had a practical drawback: in almost every other line we had to make transitions like the one from TRUE(nonempty(TRUE(b)) to TRUE(b) (see section 4 below). This made texts in AUT-68 about twice as long as the corresponding texts in AUT-QE. An example of an AUT-68 text were this drawback is manifest, in spite of an efficient abbreviation system, is presented in [3].

Nevertheless it would be a pity if AUT-68, simple and powerful as it is, would be discarded because of such a minor inconvenience. It seems worthwhile to try to get around it either by better use of the language or by slight extensions or modifications. At present, the author has no suggestion in the direction of better use of AUT-68, and concentrates on extension of that language. The extensions produce something that gets remarkably close to J. Zucker's AUT-Π (see[6]).

2. Notation. We use the colon for typing. If A has category B we write A : B (in [5] this was written as A E B). And abstractors are written as [x : A] (instead of [x,A] in older papers).

3. Use of two different 1-expressions. AUT-68 has just one 1-expression, viz. type. If f : F : type then there are two interpretations : In the "object - interpretation" f stands for an object, F for a type of objects (example f is the number 8, F stands for"number"). In the "proof-interpretation" f stands for a proof, F stands for the thing that is proved by f.

It does not do much harm to use different 1-expressions, viz. type and prop, where f : F : type is used in cases of object-interpretation, and f : F : prop is used in cases of proof-interpretation. We of course have to forbid that an F with F : type is substituted for a prop-variable (i.e. an x with x : prop) and vice versa. It has the disadvantage that we can no longer write things which are intended for both interpretations simultaneously, and, on the other hand, the advantage that we can write axioms for the proof - interpretation case without stating things on objects that we did not intend to state for objects, and vice versa, (e.g. it is possible to write the double negation axiom with prop in such a way that it turns into the axiom for Hilbert's epsilon operator if prop is replaced by type). On the whole, the advantage wins from the disadvantage, even so much so, that we are tempted to look around for use of further 1-expressions, to be used for the description of things we would not like to refer to as either objects or proofs. We shall not do this in this note.

The use of prop and type is decribed in full in [5]. Since its influence on the language properties is so minuscule, no attempt has been made to invent separate names for the type & prop-versions of AUT-68 and AUT-QE.

4. Use of "bool". There is a very strong relation between a proposition and the class of its proofs. We can even decide never to mention propositions, but just their proof classes. (This is succesful in AUT-QE, where 1-expression like [x : A] prop can be used for the typing of predicates, a possibility we do not have in AUT-68).

But if we want or have to make use of names for propositions along with names for their proof classes, we have to have a way to pass from one to the other and vice versa. First we create a 2-expression "bool", and for every b with b : bool we create its proof class. Let us call that one TRUE(b) (a better name might be PROOF(b)), so TRUE(b) : prop. Conversely, for every B with B : prop we have to have a proposition of which B is the proof class. Let us call it nonempty (B) (a slightly more suggestive name would be "nonemptyness (B)"). If we have a proof for the nonemptyness of the proof class B we want to be able to say that we have a proof for B and vice versa. This gives rise to the following setup (the numbers on the left will be used for reference in this note).

1	–	bool	: =	PN	:	<u>type</u>
2	–	b	: =	—	:	bool
3	b	TRUE	: =	PN	:	<u>prop</u>
4	–	B	: =	—	:	<u>prop</u>
5	B	nonempty	: =	PN	:	bool
6	B	u	: =	—	:	B
7	u	ax 1	: =	PN	:	TRUE(nonempty(B))
8	B	v	: =	—	:	TRUE(nonempty(B))
9	v	ax 2	: =	PN	:	B

For every logical entity we get <u>two</u> identifiers by means of which it is operated. As an example, we take the conjunction of two propositions. If b_1: : bool, b_2 : bool we can define and(b_1,b_2) (and(b_1,b_2) : bool) and next introduce AND(b_1,b_2), defined as TRUE(and(b_1,b_2)). Or, we might start from AND(b_1, b_2) : <u>prop</u> and introduce and(b_1,b_2) as nonempty(AND(b_1,b_2)). For various reasons, we want to have both forms available, and to switch from "and" to "AND" whenever necessary. It is even worse: we also want to have the arguments in both forms: If B_1 : <u>prop</u> and B_2 : <u>prop</u> we want to define a conjunction of B_1 and B_2, which can be done by

$$\mathrm{CONJ}(B_1,B_2) := \mathrm{AND}(\mathrm{nonempty}(B_1),\mathrm{nonempty}(B_2)),$$

and we want to get to and fro from CONJ to AND.

5. <u>An easy notation</u>. The following notation for "TRUE" and "nonempty" is easy to handle in handwritten texts. If an expression K satisfies K : <u>prop</u> we write $\underleftarrow{K}$ instead of nonempty(K), if L satisfies L : bool we write $\underrightarrow{L}$ instead of TRUE(L). This saves identifiers, and makes texts more readable. We can write CONJ(B_1,B_2) as $\underrightarrow{\mathrm{and}(\underleftarrow{B_1},\underleftarrow{B_2})}$, e.c. The axioms ax1 and ax2 of section 4 express that if we have something in B(where B : <u>prop</u>) then we have something in $\underrightarrow{\underleftarrow{B}}$, and vice versa.

Note: The $\rightarrow$ and $\leftarrow$ are suggested by a certain format to write AUTOMATH in, where we write in three columns, and the expressions of degree i are written in the i-th coulmn from the right (i=1,2,3; see[4]).

6. A language extension. Let us extend the language by taking "bool", "TRUE", "nonempty" (or $\rightarrow$ and $\leftarrow$) as primitive constants of the language, to be operated in the way we intend. This language extension is not very revolutionary, since every correct book in that language becomes a correct book in AUT-68 just by adding lines 1,2,3,4,5 (of section 4) at the top.

But let us extend the language a little further by adding new cases of definitional equivalence. We proclaim that

(i) If B : prop then B is definitionally equivalent to $\underset{\overset{\leftarrow}{\rightarrow}}{B}$,

(ii) if b : bool then b is definitionally equivalent to $\underset{\overset{\rightarrow}{\leftarrow}}{b}$.

Let us denote this extended language by AUT 68^{+}.

First we remark that the identification of B with $\underset{\overset{\leftarrow}{\rightarrow}}{B}$ produces about the same things as the axioms ax1 and ax2 of section 4. It is just easier. In AUT 68 we write, e.g.,

$$\begin{aligned} p &:= \;\ldots &&: B \\ q &:= \mathrm{ax1}(B,p) &&: \underset{\overset{\leftarrow}{\rightarrow}}{B} \end{aligned}$$

and

$$\begin{aligned} r &:= \;\ldots &&: \underset{\overset{\leftarrow}{\rightarrow}}{B} \\ q &:= \mathrm{ax2}(B,r) &&: B \end{aligned}$$

whereas we write in AUT 68^{+}

$$\begin{aligned} p &:= \;\ldots &&: B \\ q &:= \;p &&: \underset{\overset{\leftarrow}{\rightarrow}}{B} \end{aligned}$$

and

$$\begin{aligned} r &:= \;\ldots &&: \underset{\overset{\leftarrow}{\rightarrow}}{B} \\ s &:= \;r &&: B. \end{aligned}$$

In most cases we would not spend a line for writing q : = p or s : = r, of course.

If b is $\underset{\leftarrow}{B}$ (b : bool, B : prop) then rule (i) implies identification of $\underset{\overset{\rightarrow}{\leftarrow}}{b}$ to $\underset{\leftarrow}{B}$, i.e. to b. This does not yet imply rule (ii), for on the basis of (i) alone we cannot say that every b is definitonally equivalent to some $\underset{\leftarrow}{B}$.

That fact is a direct consequence of rule (ii). So note that, once (i) has been accepted, rule (ii) says nothing new apart from saying that every b (with b : bool) can be written as $\underset{\leftarrow}{B}$ with some B : prop.

7. Lack of symmetry. Note that the symmetry between type and prop in the type & prop-version of AUT-68 is lost in AUT-68^+. In fact we have a first order language in the sense of type and a higher order language in the sense of prop. Admittedly, we are not allowed to quantify over prop (i.e. [x : prop]is forbiden), but we can quantify over bool (even over things like [x : X][y : Y]bool) and we can get up and down from bool to prop by means of $\rightarrow$ and $\leftarrow$ (this quantification facility is present in AUT-68 too, with the axioms of section 5, but in AUT-68^+ it is much easier).

This kind of quantification has no feature like the type inclusion of AUT-QE. If, in AUT-QE, A : [x : X]prop, we may also write A : prop. In AUT-68^+ the equivalent of "[x : X]prop" is "[x : X]bool", and this is definitely not the same thing as "bool" itself.

8. Other form of AUT-68^+. Referring to the way of writing in columns (cf. the end of section 5) we might play the following trick: If S has the form $[x_1 : X_1]\ldots[x_n : X_n]$bool we agree that we shift lines like s : S one column to the right, replacing bool by prop. That is, instead of

	s	$[x_1{:}X]\ldots[x_n{:}X]$bool	type

we write

	—	s	$[x_1{:}X_1]\ldots[x_n{:}X_n]$prop

and in such case the first column has to be left open : s is not the category of anything! There is an exception : if n = 0 we admit things in the first column: Note that in AUT 68^+ s : bool implies $\underset{\rightarrow}{s}$: prop, and $\underset{\rightarrow}{s}$ can be the category of something. (In our interpretation, the things u with u : $\underset{\rightarrow}{s}$ are proofs for s).

The advantage in this formulation over the one of section 6 is that the operators $\rightarrow$ and $\leftarrow$ disappear entirely. That is, we do not only identify B and $\underset{\rightarrow}{\underset{\leftarrow}{B}}$, but also B and $\underset{\leftarrow}{B}$.

Let us call this new version AUT-68^{++}.

9. Relation with AUT-Π. Needless to say, we can express the rules of AUT 68^{++} without using AUT-68^+ as an intermediate stage. Doing so, we notice that in AUT-68^{++} a certain kind of quantification over prop-expressions is allowed.

If in AUT-68^{++} we forbid this kind of quantifications, we get a language (let us call it AUT-68^{++-}) that is weaker but still very manageable. Actually it is equivalent to a fragment of Zucker's AUT-Π[6].

10. Alternative formulation of AUT-68^{+}. Instead of shifting lines $s : [x_1 : X_1] \ldots$ $\ldots [x_n : X_n]$bool one place to the right, we might decide to get a similar effect by shifting lines u : B (with B : prop) one place to the left. That means that proofs become 4-expressions, and a 4-expression u is admitted only if cat(cat(u)) = bool where cat(bool) = type. If this formulation quantification over a 3-expression b is allowed, provided that cat(b) = bool. Now prop is no longer used, the only 1-expression is type.

References:

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

[2] N.G. de Bruijn: AUTOMATH, a language for mathematics. A series of lectures by N.G. de Bruijn, at the Séminaire de mathématiques supérieurs, Université de Montréal, June 1971. Les Presses de l'Université de Montréal, 1973. Lecture Notes prepared by B.Fawcett.

[3] N.G. de Bruijn: Some abbreviations in the input language for AUTOMATH. (Notitie 15, Department of Mathematics, T.H. Eindhoven, Apr. 1972).

[4] N.G. de Bruijn: A framework for the description of a number of numbers of the AUTOMATH family. (Internal Report, Department of Mathematics, T.H. Eindhoven, June 1974).

[5] D.T. van Daalen: A description of AUTOMATH and some of its language theory. Proceedings of the Symposium APLASM. Vol. I, ed. P.Braffort, Orsay, France (Dec. 1973).

[6] J.Zucker: Formalization of classical mathematics in AUTOMATH. Actes of the International Logic Colloquium, Clermont-Ferrand, July 1975.