A framework for the description of a number of members of the AUTOMATH family

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics

Memorandum 1974-08

Issued June, 1974

A FRAMEWORK FOR THE DESCRIPTION OF A NUMBER OF MEMBERS OF THE AUTOMATH FAMILY

by

N.G. de Bruijn

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

A framework for the description of a number of members of the AUTOMATH family

by

N.G. de Bruijn

1. Introduction. In this note we shall use a common framework in order to describe a number of members of the AUTOMATH language family.

Some of the basic features of AUTOMATH will not be repeated in this note since they have no influence on the distinctions between the various family members. We mention: the role of identifiers, block openers, context indication, syntax of expressions, PN's, the abbreviational system. We refer to [1, 2] for these things. The notion of definitional equivalence is also a common feature, except for the question what cases of beta reduction and eta reduction are to be admitted.

2. Degrees of expressions. There will be a typing relation in the set of all expressions. It is written in infix fashion: A E B, and it is said that A has type B, or that B is the type of A. (We use letters A,B,K,... as metalinguistic symbols representing expressions). The type of an expression is uniquely determined up to definitional equivalence.

There is a set of expressions for which we do not have a type; these are called 1-expressions or expressions of degree 1. For all other expressions K there is a finite sequence of typings $K \mathrel{E} K_1 \mathrel{E} \ldots \mathrel{E} K_{n-1}$ where K_{n-1} is a 1-expression.

We say that K has degree n. (So if A E B, the degree of B is 1 less than the one of A). The degree is invariant under definitional equivalence.

In order to describe the 1-expressions, we start from a set of primitive linguistic symbols. They are different from the identifiers that can be defined in an AUTOMATH book. We shall use letters τ, σ, ... for these primitives.

As 1-expressions we admit primitive symbols, as well as primitive symbols preceded by a string of abstractors, like

$$[x_1,A_1] \ldots [x_n,A_n]\tau. \tag{2.1}$$

Conditions that describe what A_i's can be admitted in order to make (2.1) "valid", have to be formulated later (section 6). These conditions will depend on the context.

The τ is called the root of (2.1); if K E ... E K_{n-1} and if the root of K_{n-1} is τ, then we also say that τ is the root of K. So every expression has as its root one of the primitive 1-symbols.

The root and the degree of an expression A will be denoted by root (A) and degree (A).

3. Mock 1-expressions and mock typings. If τ is a primitive 1-symbol, we denote, in some context, by τ_M the class of all valid 1-expressions of the form (2.1), without restrictions on n. Since we require the A_i to be valid, the class τ_M depends on the context. The class τ_M includes τ itself.

A mock 1-expression is a thing of the form

$$[y_1B_1] \ldots [y_m,B_m]\tau_M\ , \tag{3.1}$$

(possibly with m = 0). It denotes the class of all valid expressions of the form

$$[y_1,B_1] \ldots [y_m,B_m]\,[x_1,A_1] \ldots [x_n,A_n]\tau, \tag{3.2}$$

with unspecified n, $A_1,\ldots,A_n$ (n may be zero).

A mock typing is a formula of the form

$$A \quad E \quad [y_1,B_1] \ldots [y_m,B_m]\tau_M\ . \tag{3.3}$$

It is intended to express that the type of A belongs to the class (3.1). So (3.3) gives an existence statement: there exist $n,A_1,\ldots,A_n$ such that the type of A is (3.2). Since a mock 1-expression describes a class of expressions, the following usage of $\in$ and $\subset$ is obvious: If T_1,T_2 stand for abstractor strings (possibly empty) we have

$$T_1T_2\tau \ \in\ T_1\tau_M \quad , \quad T_1T_2\tau_M \ \subset\ T_1\tau_M.$$

If we have A E B and B $\in$ C or B $\subset$ C, then the transition of B to C (in order to get from A E B to A E C) is not a reduction that leads to definitional equality, like beta or eta reduction. The way we might consider it, is "sacrifice of information". As long as B is no mock expression, a formula A E B gives full information on the type of A; if A E B and B $\in$ C, then A E C gives partial information only; if C $\subset$ D then the transition from A E C to A E D can be a further loss of information.

4. Limitations on degrees. For every primitive 1-symbol τ we can, if we wish, fix an integer n > 0, to be called "degree bound", and agree that no expressions of degree exceeding n with root τ will be admitted.

5. Irrelevance degrees. For every primitive 1-symbol τ we can, if we wish, fix one or more integers n > 1, to be called irrelevance degrees, with the following agreement on definitional equivalence: if A_1 and A_2 both have

root τ and degree n (where n is an irrelevance degree), and if A_1 E B, A_2 E B then $A_1 \overset{D}{=} A_2$.

6. Admissible contexts. A context can be given by a sequence of abstractors

$$[x_1,A_1] \ldots [x_n,A_n]. \tag{6.1}$$

If B is a correct expression, and if its degree is strictly less than the degree bound of its root (so that there is no objection to having things that have B as their type) we admit to open a block y := ——— B. That is, we admit

$$[x_1A_1] \ldots [x_n,A_n][y,B] \tag{6.2}$$

as a new context.

We also admit this if B is a mock 1-expression.

7. Abstraction. Assuming that in the context (6.2) we have a typing or mock typing P(y) E Q(y), then we wish to conclude in the context (6.1) that

$$[y,B]\ P(y) \quad E \quad [y,B]\ Q(y). \tag{7.1}$$

The right to do this can be limited by means of a list of quadruples (σ,k,τ,ℓ) (where σ,τ stand for primitive 1-symbols, and k,ℓ for positive integers). If we want to know whether (7.1) is admitted, we take for σ the root of B, for k the degree of B, for τ the root of P(y), for ℓ the degree of P(y). If this quadruple is on the list, the abstraction is admitted.

In any case, we require that B is an expression, and <u>not</u> a mock 1-expression. There is no objection for Q(y) to be a mock 1-expression.

8. Application rules. We shall consider two kinds of rules.

(i) If, in a certain context, we have

$$D \quad E \quad [x,B]C(x),$$

where C(x) is either an expression or a mock 1-expression, and

$$A \quad E \quad B,$$

then we have

$$\{A\}D \quad E \quad C(A), \tag{8.1}$$

provided that the quadruple (root (A), degree (A), root (D), degree (D)) occurs on a list made for this rule.

(ii) If in a certain context we have

$$D_1 \; E \; D_2 \; E \; \ldots \; E \; D_k \; E \; [x,B]C(x),$$

where C(x) is either an expression or a mock 1-expression, and

$$A \; E \; B,$$

then we have

$$\{A\}D_1 \quad E \quad \{A\}D_2 \tag{8.2}$$

provided that the quintuple (root (A), degree (A), root (D_1), degree (D_1),k) occurs on a list made for this rule.

Remark. We can, of course, separately deal with the case that C(x) is a mock 1-expression, using lists that are different from those made for the case that C(x) is an expression. This does not seem to be very useful, however.

9. Beta reduction. The right to reduce in a certain context

$$\{A\}[x,B]\ C(x) \quad \text{to} \quad C(A),$$

(with $A \mathrel{E} B$), may be restricted to special quadruples (root (A), degree (A), root (C(x)), degree (C(x))).

10. Eta reduction. The right to reduce, in a certain context

$$[x,B]\{x\}C \quad \text{to} \quad C$$

(where C does not contain x) may be restricted to special quadruples (root (B), degree (B), root (C), degree (C)).

11. Substitution rules in connection with mock expressions. Assume we have opened a block with $x := ---\Gamma$, where Γ stands for a mock expression, and that inside the block we have $f := A(x) \mathrel{E} B(x)$ (where B(x) may be a mock expression). Then we can use f outside the block: if we have either $P \mathrel{E} K$, (with $K \in \Gamma$) or $P \mathrel{E} \Sigma$ (with $\Sigma \subset \Gamma$), then we also have $f(P) \mathrel{E} B(P)$ (and this may be again a mock typing). A complete formulation of this substitution rule has to contain the case of several variables, and has to give a more serious description of contexts than the mere phrase "outside the block". We do not go into this here, since these things are standard AUTOMATH features.

12. Primitive notions. In general, a primitive notion is something that, in a context like (6.1) is described to have type B, where B satisfies the same condition as the type of a new variable (see the beginning of section 6): degree (B) should be less than the degree bound of root (B).

One might hesitate to include the case that B is a mock 1-expression. It seems a bit funny. A mock typing $A \mathrel{E} B$ can be seen as incomplete information

about the true typing: it just says that B contains the type of A. If we introduce a primitive notion, we want to interpret it as something with a fixed meaning thoughout the rest of the book. It seems strange to leave the reader uncertain about the nature of the notion.

If one writes a theory with such mock PN's, one should realize that one leaves some freedom to those who want to build models for the theory.

These remarks are not necessarily objections, but nevertheless one may think of forbidding the case that B is a mock expression at least for some values of root (B).

13. Fitting special languages into this framework. Let us first discuss standard AUTOMATH. It has just one primitive 1-symbol τ, and the derived mock symbol τ_M is called type. The τ is not explicitely used, and neither are typings of the form A E $T\tau_M$ (where T is a non-empty abstractor string). Accordingly all PN's of degree 2 are mock PN's.

The only degrees to be admitted are 1,2,3. There are no irrelevance degrees. As far as abstraction is concerned, we only admit (with the notation of (7.1)) degree (B) = 2, degree (P(y)) = 3.

The application rule (8.1) is restricted to degree (A) = 3, degree (D) = 3. The rule (8.2) is absent. Beta reduction and eta reduction require (notation of section 9 and 10) degree (A) = 3, degree C(x) = 3, degree (C) = 3. Since these are the only situations that ever occur, we might as well say that there are no restrictions on the degrees.

In AUT-QE we again have a single τ that remains anonymous, but, in contrast to AUTOMATH, $T\tau_M$'s can be used. The PN's can have type $T\tau_M$. Again, the only degrees to be admitted are 1,2,3 and there are no irrelevance degrees. Abstraction is admitted if (with the notation of (7.1)) degree (B) = 2, degree (P(y)) = 2 or 3. For the application rule (8.1) we require

degree (A) = 3, degree (D) = 2, and for (8.2) we require degree (A) = 3, degree (D_1) = 3, k = 2. In beta and eta reduction there are no restrictions on the degrees.

In AUT-SL(see [3],[4]) we have a single type τ, and no use is made of mock 1-expressions. There is no bound on the degrees, and there are no irrelevance degrees. There are no restrictions in (7.1) on the degrees of either B or P(y) (though the case degree(P(y)) = 1 will not occur). There are no restrictions on degrees in (8.1) (though degree (D) = 1 will not occur), and in (8.2) there is no restriction on degree (A), degree (D_1) or k. There is no restriction on the degrees in beta and eta reduction.

Since in AUT-SL abstraction is universally possible, PN's can be written as block openers, all lines can be reduced to zero context, and every result of the book can be represented by means of a book consisting of a single line. These unessential features give AUT-SL an appearance slightly different from the other members of the AUTOMATH family.

There have been experiments with the use of more than one primitive 1-symbol in AUTOMATH and AUT-QE. The symbols used were type and prop. The interpretation of things like A E B E type and C E D E prop is that A is an object, B is a class to which A belongs, D is a proposition and C is a proof. Keeping type and prop apart, has the effect that axioms expressed for all propositions do not automatically carry over to all classes. Moreover, it will be possible to control abstraction, mock expressions, etc., by rules that are not the same for the cases type and prop. This is attractive if we want to translate existing formal systems into AUTOMATH-like languages: many such systems treat classes in a way entirely different from the treatment of propositions.

One may also think of extra primitive 1-symbols for the treatment of further kinds of constructions, like geometric constructions, computer programs. If languages will ever claim to realize Leibniz's idea of a

universal language for science, they may need ways for directly discussing situations in fields like physics, rational mechanics, chemistry, without the use of an intermediate mathematical model. This breaks with the usual idea to formalize the mathematical model only, leaving the relation between that model and the things it describes to intuition.

The matter of irrelevance degrees (section 5) did not come up yet in the languages mentioned thus far. (It first came up in the fall of 1973 in an unpublished language called AUT-4). The use for which this feature has been primarily intended is "irrelevance of proofs". We shall explain this in some detail.

Assume that we have something that is defined only if a certain condition on a number of objects is satisfied. To take an example: if s is a sequence of reals, then lim s (the limit of the sequence) is defined only if a certain condition (let us say the condition that s is a fundamental sequence) holds.

In AUTOMATH the definition can be written as follows

```
s : =  ----------  [n,nat]real
       u :=  ----------  fund(s)
       lim :=    K          real
```

Here fund(s) can be interpreted as the(possibly empty) class of all proofs that show s to be a fundamental sequence, and K stands for the expression that defines the limit.

Outside this block we can refer to the limit of a sequence A only if we have some proof B that proves A to be fundamental, and the limit is lim(A,B). If we have two different proofs B_1,B_2, then we get two different expressions lim(A,B_1) and lim(A,B_2) which need not be definitionally

equivalent. We may try to prove that the two expressions are equal in the sense of some equality notion introduced in the book, but that is not the same thing.

The situation is much more comfortable if we have an irrelevance degree rule. Let us start with two primitive symbols (or rather mock symbols) type and prop.

For both we admit degrees 1,2,3, and for prop we take 3 as irrelevance degree. We now have, in the above example A E [n,nat]real E type and B_1 E fund(A) E prop, B_2 E fund(A) E prop. Since B_1 and B_2 have the same type, and since 3 is an irrelevance degree for prop, the expressions $\lim(A,B_1)$ and $\lim(A,B_2)$ are definitionally equivalent.

References

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

[2] 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 Montreal, 1971.

[3] N.G. de Bruijn, "AUT-SL, a single line version of AUTOMATH", Technological University Eindhoven, Department of Mathematics, Internal Report, notitie 22, 1971.

[4] R.P. Nederpelt, "Strong normalization in a typed lambda calculus with lambda structured types", Doctoral Thesis, Technological University, Eindhoven , 1973.