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

de Bruijn, N.G.

Published: 01/01/1974

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 14. Jan. 2019
A FRAMEWORK FOR THE DESCRIPTION OF A NUMBER OF MEMBERS
OF THE AUTOMATH FAMILY

by

N.G. de Bruijn
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: $A \in B$, and it is said that $A$ has type $B$, or that $B$ is the type of $A$. (We use letters $A, B, K, \ldots$ 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 l-expressions or expressions of degree 1. For all other expressions $K$ there is a finite sequence of typings $K \in K_1 \in \ldots \in K_{n-1}$ where $K_{n-1}$ is a l-expression.
We say that $K$ has degree $n$. (So if $A \in B$, the degree of $B$ is 1 less than the one of $A$). The degree is invariant under definitional equivalence.

In order to describe the $l$-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 $\tau, \sigma, \ldots$ for these primitives.

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

$$[x_1, A_1] \ldots [x_n, A_n] \tau.$$  \hspace{1cm} (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 $\tau$ is called the root of (2.1); if $K \in \ldots \in K_{n-1}$ and if the root of $K_{n-1}$ is $\tau$, then we also say that $\tau$ is the root of $K$. So every expression has as its root one of the primitive $l$-symbols.

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

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

A mock $l$-expression is a thing of the form

$$[y_1, B_1] \ldots [y_m, B_m] \tau_M,$$  \hspace{1cm} (3.1)

(possibly with $m = 0$). It denotes the class of all valid expressions of the form
\[ [y_1, B_1] \ldots [y_m, B_m] [x_1, A_1] \ldots [x_n, A_n], \tag{3.2} \]

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

A mock typing is a formula of the form

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

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 \( \lambda \)-expression describes a class of expressions, the following usage of \( \varepsilon \) and \( c \) is obvious: If \( T_1, T_2 \) stand for abstracter strings (possibly empty) we have

\[ T_1 T_2 \varepsilon \in T_1 \tau_M, \quad T_1 T_2 \varepsilon = T_1 \tau_M'. \]

If we have \( A \in B \) and \( B \in C \) or \( B \subset C \), then the transition of \( B \) to \( C \) (in order to get from \( A \in B \) to \( A \in C \)) 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 \( A \in B \) gives full information on the type of \( A \); if \( A \in B \) and \( B \subset C \), then \( A \in C \) gives partial information only; if \( C \subset D \) then the transition from \( A \in C \) to \( A \in D \) can be a further loss of information.

4. Limitations on degrees. For every primitive \( \lambda \)-symbol \( \tau \) 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 \( \tau \) will be admitted.

5. Irrelevance degrees. For every primitive \( \lambda \)-symbol \( \tau \) 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 \( \tau \) and degree \( n \) (where \( n \) is an irrelevance degree), and if \( A_1 \in B \), \( A_2 \in B \) then \( A_1 \stackrel{D}{=} A_2 \).

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

\[
[x_1, A_1] \ldots [x_n, A_n].
\]  

(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 \( y := \_\ldots B \). That is, we admit

\[
[x_1, A_1] \ldots [x_n, A_n][y, B]
\]

(6.2)
as a new context.

We also admit this if \( B \) is a mock \( I \)-expression.

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

\[
[y, B] P(y) \in [y, B] Q(y).
\]

(7.1)

The right to do this can be limited by means of a list of quadruples \((\sigma, k, \tau, \iota)\) (where \( \sigma, \tau \) stand for primitive \( I \)-symbols, and \( k, \iota \) for positive integers). If we want to know whether (7.1) is admitted, we take for \( \sigma \) the root of \( B \), for \( k \) the degree of \( B \), for \( \tau \) the root of \( P(y) \), for \( \iota \) 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 not a mock \( I \)-expression. There is no objection for \( Q(y) \) to be a mock \( I \)-expression.
8. Application rules. We shall consider two kinds of rules.

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

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

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

\[ A \ E \ B, \]

then we have

\[ \{A\}D \ E \ C(A), \quad (8.1) \]

provided that the quadruple \((\text{root (A)}, \text{degree (A)}, \text{root (D)}, \text{degree (D)})\) 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

\[ \{A\}D_1 \ E \ \{A\}D_2 \quad (8.2) \]

provided that the quintuple \((\text{root (A)}, \text{degree (A)}, \text{root (D_1)}, \text{degree (D_1)}, k)\) 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

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

(with \( A \in B \)), 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

\[ [x,B][x]C \to C \]

(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 \( \Gamma \) stands for a mock expression, and
that inside the block we have \( f := A(x) \in B(x) \) (where \( B(x) \) may be a mock
expression). Then we can use \( f \) outside the block: if we have either \( P \in K \),
(with \( K \in \Gamma \)) or \( P \in \Sigma \) (with \( \Sigma \subset \Gamma \)), then we also have \( f(P) \in B(P) \) (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):
dergree (B) should be less than the degree bound of root (B).

One might hesitate to include the case that \( B \) is a mock \( l \)-expression. It
seems a bit funny. A mock typing \( A \in B \) 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 throughout 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 \( \lambda \)-symbol \( \tau \), and the derived mock symbol \( \tau_M \) is called type. The \( \tau \) is not explicitly used, and neither are typings of the form \( A \in T \tau_M \) (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 \( \tau \) that remains anonymous, but, in contrast to AUTOMATH, \( \tau_M \)'s can be used. The PN's can have type \( \tau_M \). 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 \(\tau\), and no use is made of
mock \(I\)-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 \(\text{degree}(P(y)) = 1\) will not occur).
There are no restrictions on degrees in (8.1) (though \(\text{degree}(D) = 1\) 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
\(I\)-symbol in AUTOMATH and AUT-QE. The symbols used were \(\text{type}\) and \(\text{prop}\).
The interpretation of things like \(A \in B \in \text{type}\) and \(C \in D \in \text{prop}\) is that
\(A\) is an object, \(B\) is a class to which \(A\) belongs, \(D\) is a proposition and \(C\)
is a proof. Keeping \(\text{type}\) and \(\text{prop}\) 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 \(\text{type}\) and \(\text{prop}\). This is attrac-
tive 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 \(I\)-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

\[
\begin{align*}
s & := \text{[n,nat]real} \\
\lim & := \text{fun(s)} \\
\end{align*}
\]

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 $A \in [\text{n,nat}]\text{real} \ E \text{type}$ and $B_1 \in \text{fund}(A) E \text{prop}, B_2 \in \text{fund}(A) E \text{prop}$. Since $B_1$ and $B_2$ have the same type, and since 3 is an irrelevance degree for prop, the expressions $\text{lim}(A,B_1)$ and $\text{lim}(A,B_2)$ are definitionally equivalent.
References


