Technological University Eindhoven,
The Netherlands.
Department of Mathematics.

A normal form theorem in a λ calculus with types

by L.S. van Benthem Jutting

In the previous "Tagung" R.P. Nederpelt has given a description of AUTOMATH ([1], [2]). It has been long conjectured that every expression in AUTOMATH has a normal form. An unpublished proof of this has been given by L.E. Fleischhacker [3]. Here a proof is presented that in a λ-calculus closely resembling AUTOMATH every correct expression has a normal form. The proof proceeds along the lines pointed out by Fleischhacker and uses a norm which is due to Nederpelt.

The importance of this theorem is that it makes it possible for us to decide whether two expressions are "equal". In fact, together with the Church-Rosser theorem (see Curry-Feys, [4]) we may deduce that two expressions are "equal" iff they have the same normal form. This helps in proving that correctness of AUTOMATH expressions is decidable.

1. Definition of the language

We will give here only a very loose definition. A strict definition may be found in [5].

We discern constants a, b, c, ..., variables x, y, z, ..., the symbol T and various brackets as primitive symbols. For the sake of clarity we will use below also other constants like 1, s and N.

Expressions are defined by:

a constant, a variable, the symbol T are expressions;

if A and B are expressions then (A)B and [x,A]B are expressions.

Intuitively expressions may be thought of as denoting objects: (A)B denotes the value of the function B for the argument A; [x,A]B denotes the function associating to every x in the domain A the value B (which may depend upon x). We will call x bound in [x,A]B.

We shall discern 3-expressions, 2-expressions and 1-expressions. Intuitively 3-expressions denote "mathematical objects", e.g. the natural number one is denoted by the expression 1, the successor-function in the natural numbers may be denoted by s, the natural number two, being the successor of one, is then denoted by (1)s.

2-expressions denote "classes" to which mathematical objects belong, e.g. the set of natural numbers, denoted by N, or the set of all function-

mappings $\mathbb{N}$ into $\mathbb{N}$, denoted by $[x,\mathbb{N}]\mathbb{N}$.

1-expressions denote "superclasses" to which classes belong, e.g. the superclass of all classes, denoted by T, or the superclass of the classes of mappings of $\mathbb{N}$ into some other class, denoted by $[x,\mathbb{N}]T$. Syntactically 1-expressions are those expressions which have T as their last symbol.

Now every mathematical object belongs, in our conception, to exactly one class and every class to exactly one superclass. This induces a function γ, called type, mapping 3-expressions into 2-expressions and 2-expressions into 1-expressions. E.g. $\gamma(1) \equiv \mathbb{N}$, $\gamma(s) \equiv [x,\mathbb{N}]\mathbb{N}$, $\gamma(\mathbb{N}) \equiv T$ etc. It follows that we must discern between the natural number one, with $\gamma(1) \equiv \mathbb{N}$, and the real number one, denoted by 1^* with $\gamma(1^*) \equiv \mathbb{R}$. It will be clear now that an expressions A is either a 1-expression, or A is a 2-expression and then $\gamma(A)$ is a 1-expression or A is a 3-expression and then $\gamma(A)$ is a 2-expression and $\gamma(\gamma(A))$ a 1-expression.

The type γ must be thought of as defined on a finite number of constants. It may be extended to a new constant a by defining $\gamma(a)$ as a certain 2-expression or 1-expression which contains only constants defined before a. In this case a must be thought of as denoting a definite object of the class or superclass denoted by $\gamma(a)$. We will say that a is a defined constant.

On bound variables the type γ is defined, too: in $[x,A]B$ the variable x, which might occur free in B, has type $\gamma(x) \equiv A$. Hence A must be a 2-expression or a 1-expression (otherwise the expression $[x,A]B$ is incorrect). On composite expressions γ may be defined recursively.

We now give a notation for substitution: the result of substituting the expression A for the variable x in the expression B is denoted by $(x := A)B$. A definition of substitution we will omit here.
The intuitive meaning of $\{A\}B$ and $[x,A]B$ leads us to a definition of reduction as follows:

α) $[x,A]B \geq [y,A](x := y)B$ if y is not free in B.

β) $\{A\}[x,B]C \geq (x := A)C$.

η) $[x,A]\{x\}B \geq B$ if x is not free in B.

Intuitively the expressions to the right and to the left of $\geq$ denote the same objects. We extend the relation $\geq$ to a monotone quasi order on all expressions, i.e. if $A \geq C$ and $B \geq D$ then $\{A\}B \geq \{C\}D$ and $[x,A]B \geq [x,C]D$.

Now there are rules according to which it may be decided whether an expression is correct. One of these was mentioned above: in $[x,A]B$, A should be either a 2-expression or a 1-expression. The main ideas are:

a) A correct expression does not contain free variables or undefined constants.

b) $\{A\}B$ is only correct if B denotes a function and A belongs to the domain of that function (i.e. A is not a 1-expression and $\gamma(A)$ is the domain of B).

c) The constants should be defined in due order, and for every defined constant a, $\gamma(a)$ should be correct with respect to the constants defined before.

2. The normal form theorem

We say that A is in normal form (in n.f.) if neither A nor any subexpression of A is β or η reducible. It follows that if A is in normal form then

$$A \equiv [x_1,B_1][x_2,B_2]\ldots[x_m,B_m]\{D_1\}\ldots\{D_n\}p$$

where n, m are non-negative integers, p denotes a constant, a variable or the symbol T and $B_1,\ldots B_m$, $D_1,\ldots D_n$ are in n.f. We say that A has a normal form if B in n.f. exists such that $A \triangleright B$. We now introduce the norm τ on expressions as follows

$\tau(a) = \tau(\gamma(a))$ for all defined constants a.

$\tau(b) = 0$ for all undefined constants b.

$\tau(x) = \tau(A)$ if x is bound by $[x,A]$.

$\tau(y) = 0$ if y is free.

$$\tau(\{A\}B) = \begin{cases} p & \text{if } \tau(A) \neq 0 \text{ and } \tau(B) = [\tau(A)]P \text{ for a certain symbol-string P} \\ 0 & \text{otherwise} \end{cases}$$

$$\tau([x,A]B) = \begin{cases} [\tau(A)]\tau(B) & \text{if } \tau(A) \neq 0 \text{ and } \tau(B) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

A strong point of this norm is that is is invariant under substitution and reduction:

Theorem 1: If $\tau(B) \neq 0$ and $\tau(A) = \tau(x) \neq 0$ then $\tau((x := A)B) = \tau(B)$

The proof is easy when substitution is well defined.

Theorem 2: If $\tau(A) \neq 0$ and $A \triangleright B$ then $\tau(B) = \tau(A)$.

We will prove this for β reduction:
Suppose $A \equiv \{C\}[x,D]E$ and $B \equiv (x := C)E$.
As $\tau(A) \neq 0$ we know that $\tau(C) \neq 0$ and $\tau([x,D]E) = [\tau(C)]\tau(A)$.
Hence $\tau([x,D]E) \neq 0$ and it follows that $\tau([x,D]E) = [\tau(D)]\tau(E)$.
It follows that $\tau(D) = \tau(C)$ and $\tau(E) = \tau(A)$. Moreover $\tau(x) = \tau(D)$ because x is bound by $[x,D]$, hence $\tau(x) = \tau(D) = \tau(C)$. Therefore, by theorem 1, $\tau(B) = \tau((x := C)E) = \tau(E) = \tau(A)$.

The next theorem is the crucial part in our proof.

Theorem 3: If A is in n.f. with $\tau(A) = \tau(x) \neq 0$ and B is in n.f. with $\tau(B) \neq$, then C in n.f. exists such that $(x := A)B \geq C$.

The proof is complicated and proceeds by double induction:

I) with respect to the length of $\tau(A)$,

II) with respect to the length of B.

The difficulty lies in the case when $B \equiv \{D\}x$, because then, by substituting A for x in B, an expression is obtained which is in general not in normal form.

The next theorem is an easy consequence of theorem 3.

Theorem 4: If $\tau(A) \neq 0$ then A has a normal form.

We now state

Theorem 5: If A is correct then $\tau(A) \neq 0$.

From theorem 4 and 5 now follows

Theorem 6: If A is correct then A has a normal form.

References

[1] de Bruijn, N.G., The mathematical language AUTOMATH, its usage and some of its extensions. Proc. Symp. on Automatic Demonstration. (IRIA, Versailles, December 1968), Springer Lecture Notes Series, Berlin, 1969.

[2] Nederpelt, R.P., AUTOMATH, a language for checking mathematics with a computer, Internal Report, Technological University Eindhoven, Department of Mathematics, 1970.

[3] Fleischhacker, L.E., Private communication, 1970.

[4] Curry, H.B., Feys, R., Combinatory Logic I, Chapter 4. North Holland Publ. Comp. Amsterdam, 1958.

[5] van Benthem Jutting, L.S. On normal forms in AUTOMATH,
Internal Report, Technological University
Eindhoven, Department of Mathematics, 1971.