A namefree lambda calculus with formulas involving symbols that represent reference transforming mappings

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics

A namefree lambda calculus with formulas involving symbols that represent reference transforming mappings.

by

N.G. de Bruijn.

Memorandum 1977-10.
Issued August 1977.

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

A namefree lambda calculus with formulas involving symbols that represent reference transforming mappings.

by N.G. de Bruijn.

1. Introduction on name-carrying lambda calculus. In ordinary lambda calculus we use names both for free and for bound variables. Let us present an example that explains what kind of expressions we are after: apart from names for variables we have names for constants that suggest function symbols too. We may have introduced an expression in two variables x and y, and have abbreviated it to f(x,y) (now f is the "constant" we mentioned). Now $\lambda_x f(x,y)$ is a lambda expression. Its interpretation is: the function that attaches to every x the value f(x,y).

Now y is a free variable and x a bound variable in the expression $\lambda_x f(x,y)$.

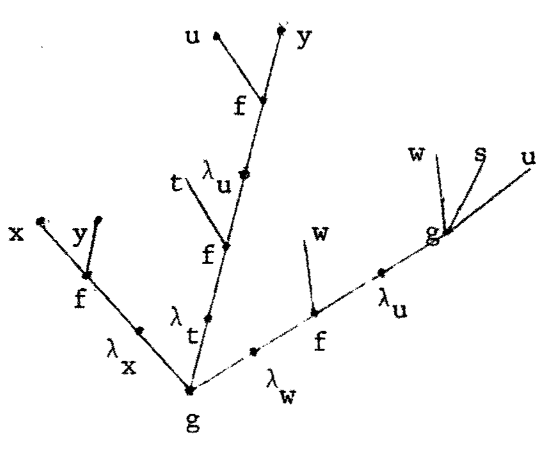
We can, of course, also write more complex lambda expressions like

$$g(\lambda_x f(x,y),\ \lambda_t f(t,\lambda_u f(u,y)),\ \lambda_w f(w,\lambda_u g(w,s,u))). \tag{1.1}$$

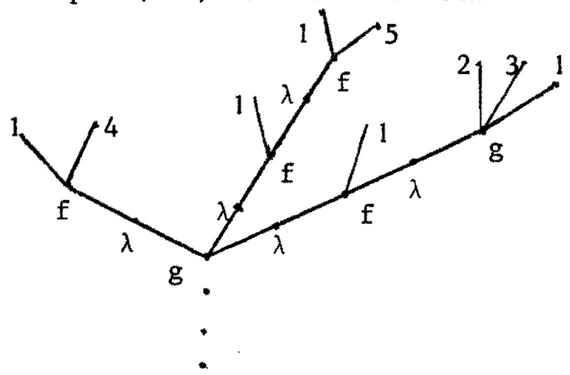
In this example the free variables are y and s.

Usual lambda calculus has a notation (in the form of concatenation) for "application" that intends to express "the value of the function y at the point x". We do not need a special notation for this, because we can devote a special constant A to this purpose, and write that value as A(x,y). Now so-called beta-reduction is a kind of elimination of such an A, like the passage from $A(\lambda_x(f(x,y)),g(t))$ to f(g(t),y). The latter two formulas are not considered to be equal (in spite of their common interpretation). On the other hand, the difference between $\lambda_x f(x,y)$ and $\lambda_u f(u,y)$ is much less essential. The desire to identify them lies at the root of namefree lambda calculus.

The kind of name-carrying lambda calculus described above is exactly the same as in [1]. We close this section with the tree interpretation of the expression (1.1):



2. Introduction on namefree lambda calculus. In [1] we explained a notation for lambda expressions where all occurrences of free and bound variables are replaced by positive integers that indicate their reference depth. The system is easily demonstrated at the example (1.1) in the tree form



with s,z,y,... as list of free variables.

The dots below the tree are unessential, but suggestive to the term "reference depth", if they are interpreted as being tied to s,z,y,... (the upper one refers to s, the middle one to z, the lower one to y; the fact that z is never referred to in the formula does not bother us).

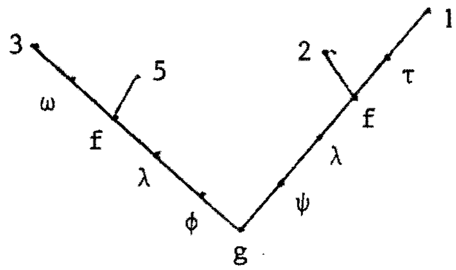
The idea is that an integer k at an end-point refers to the k-th lambda we meet when travelling from that end-point to the root of the tree; if there are only j lambdas on that path, with $j < k$, then the k at the endpoint refers to the $(k-j)$-th entry of the list of free varibales.

As a preparation to what follows, we express the above correspondence like this. We start at an endpoint and want to know what variable the number refers to. Now we descend the tree, taking the number along, subtracting 1 each time we pass some λ. If this subtraction leads to the value 0, we do not go any further; we have located the right lambda. Of course we act as if the free variables s,z,y, are tied to underground lambdas.

The tree at the beginning of this section did not show any names attached to the lambdas. We can assign arbitrarily names (different from the free variables) for these bound variables. There is a safe, "conservative", system where it is required that all these names are different. The "liberal" system, on the other hand, only requires that lambdas get different names if they are hierarchically related:

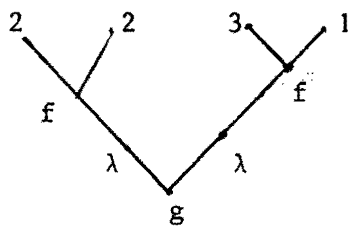
if one lambda lies on the path down from another lambda to the root, then the two have to get different names.

3. Trees with symbols that represent mappings. We shall now describe a new kind of namefree trees where at some places in the tree we have a symbol denoting some mapping of $\mathbb{N}$ into $\mathbb{N}$ ($\mathbb{N} = \{1,2,3,\ldots\}$). We shall use these more complicated trees for the same purpose as the trees in section 1. What matters is, to describe to which lambda a natural number at an end-point refers. What we intend to do will be clear from an example. The letters ω,ϕ,τ,ψ denote mappings of $\mathbb{N}$ into $\mathbb{N}$.



(3.1)

If we want to know what an integer refers to we descend the tree; again we subtract 1 if we pass a λ, but if we pass one of the letters representing a mapping we do something different: we apply that mapping to our number. So the 3 in the upper left corner refers to the left-hand λ if $\omega(3)=1$. If $\omega(3) > 1$ it refers to $\phi(\omega(3)-1)$-th free variable. As an excercise the reader may verify that if $\omega(3) = 5$, $\phi(4) = 1$, $\tau(1) = 1$, $\psi(1) = 2$, then this tree corresponds to the same references from end-points to lambdas or free variables as the following one:



(3.2)

We shall say that (3.2) is the reduced form of (1).In namefree formula notation (3.1) is represented by

$$g(\phi\, \lambda\, f(\omega 3,5),\quad \psi\lambda f(2,\tau 1)) \qquad (3.3)$$

and (3.2) by

$$g(\lambda f(2,2),\ \lambda f(3,1)). \qquad (3.4)$$

The motivation for studying the tree coding of the type (3.3) is that operations like substitution are easier described in terms of these than in terms of the mapping free codes like (3.4). This may hold both for language theory and for automatic formula manipulation. Getting rid of the mappings can be postponed until we need it; it is relatively easy.

4. <u>Metalinguistic notation</u>. In [1] our way to describe linguistic operations was based on the system used in Backus' normal form. In simple cases this is quite feasable, but in more complex situations it can no longer be maintained. In the present note we prefer the system used in the theory of context-free languages,

where linguistic entities like words are treated as mathematical objects, referred to by names or more complicated expressions, and never appear themselves in the language that discusses them. We use "combs" for indicating the concatenation of words: if p and q denote words, then $\boxed{p \mid q}$ denotes the word we get by putting the second word directly after the first one. And we use symbols like $\boxed{P \mid q}$ for indicating $\{\boxed{p \mid q} \mid p \in P\}$.

Another notation we introduce is this: if p denotes a letter, then $\sigma(p)$ denotes the atom(= one-letter word) consisting of that one letter. Here we leave it at these brfef indications: the notational system was more extensively expressed in a companion note [2].

5. The sets Z_o and Z. In order to give a preliminary idea we state that Z will consist of all strings of the type (3.3) (with a restriction on the constants) and Z_o will be the subset consisting of the strings of type (3.4). (The elements of Z_o were called NF-expressions in [1]).

As before, $\mathbb{N} = \{1,2,3,\ldots\}$. And Γ will denote the set of all mappings of $\mathbb{N}$ into $\mathbb{N}$.

We consider a set A which is the union of four disjoint sets

$$A = B \cup \mathbb{N} \cup \Gamma \cup R. \tag{5.1}$$

The set R has four distinct elements r_1, r_2, r_3, r_4. The corresponding atoms $\sigma(r_1)$, $\sigma(r_2)$, $\sigma(r_3)$, $\sigma(r_4)$ will be called the lambda, the opening parenthesis, the comma, the closing parenthesis. We write $\sigma(r_1) = \lambda$, and for $\sigma(r_2), \sigma(r_3), \sigma(r_4)$ we shall use the signs that correspond to the names just given, at least as long as they occur under combs (for otherwise there can be a danger of confusion). So we can write

$$\boxed{\sigma(r_1) \mid \sigma(r_2) \mid \sigma(r_3) \mid \sigma(r_4)} = \boxed{\lambda \mid (\mid , \mid)}.$$

We now define Z as the minimal solution of the equation

$$Z = C \cup \boxed{C \mid (\mid \mathrm{str}(Z, \boxed{,}) \mid)} \cup N \cup \boxed{\lambda \mid Z} \cup \boxed{\Phi \mid Z} \tag{5.2}$$

where $\mathrm{str}(Z, \boxed{,})$ is the set of all strings we get by writing down a number of elements of Z separated by commas (see [2]); C, N and Φ are the set of atoms produced by B, $\mathbb{N}$ and Γ, respectively:

$$C = \sigma(B), \; N = \sigma(\mathbb{N}), \; \Phi = \sigma(\Gamma) \tag{5.3}$$

where, of course, $\sigma(X)$ is defined as $\{\sigma(x) \mid x \in X\}$.

The subset Z_o of Z can be defined as the minimal solution of

$$Z_o = C \cup \boxed{C \mid (\mid \boxed{str(Z_o, \boxed{,}\,)} \mid)} \cup N \cup \boxed{\lambda \mid \boxed{Z_o}}. \qquad (5.4)$$

We shall abbreviate

$$Z' = str(Z, \boxed{,}\,) \quad , \quad Z'_o = str(Z_o, \boxed{,}\,).$$

6. <u>Substitution</u>. Let Ω be a mapping of $\mathbb{N}$ into Z, and let z be an element of Z'. We want to define subst(Ω,z). Its interpretation for the special case that $z \in Z$ is as follows. Attach to z the free variable list $x_1, x_2, \ldots$, and to each one of $\Omega(1), \Omega(2), \ldots$ the variable list $y_1, y_2, \ldots$. Now we substitute into the name-carrying form of z, for each x_i, the name-carrying form of $\Omega(i)$. What we get is an expression with free variables $y_1, y_2, \ldots$, and the namefree form of this will be subst(Ω,z). If z is a string, $z \in Z'$, then the substitution is effected in every entry of the string separately.

From now on we concentrate on what happens in Z' and Z, and we do not study the interpretations. (They will stay on the back of our mind, of course).

We define subst(Ω,z) for all $z \in Z'$ by recursion on (5.2). To that end it suffices to define (note the uniqueness of parsing the elements of Z'):

(i) if $z = \boxed{z_1 \mid , \mid z_2}$ with $z_1 \in Z'$, $z_2 \in Z$ then

$$\text{subst}(\Omega,z) = \boxed{\text{subst}(\Omega,z_1) \mid , \mid \text{subst}(\Omega,z_2)} ,$$

(ii) if $z \in C$ then subst(Ω, z) = z,

(iii) if $z = \boxed{c \mid (\mid z_1 \mid)}$ with $c \in C$, $z_1 \in Z'$ then

$$\text{subst}(\Omega,z) = \boxed{c \mid (\mid \text{subst}(\Omega,z_1) \mid)},$$

(iv) if $z = \sigma(n)$ for some $n \in \mathbb{N}$ then subst(Ω,z) = $\Omega(n)$,

(v) if $z = \boxed{\lambda \mid z_1}$ with $z_1 \in Z$ then

$$\text{subst}(\Omega,z) = \boxed{\lambda \mid \text{subst}(\Omega^*,z_1)},$$

where Ω^* is the mapping defined by

$$\Omega^*(1) = \sigma(1),\ \Omega^*(k) = \boxed{\sigma(\gamma) \mid \Omega(k-1)} \qquad (k=2,3,\ldots)$$

with γ defined by $\gamma(k) = k+1$ $(k=1,2,3,\ldots)$,

(vi) if $z = \boxed{\sigma(\phi) \mid z_1}$ with $\phi \in \Gamma$, $z_1 \in Z$ then

$$\text{subst}(\Omega,z) = \text{subst}(\Omega\phi,z_1)$$

where $\Omega\phi$ is defined by $(\Omega\phi)(k) = \Omega(\phi(k))$ for all $k \in \mathbb{N}$.

7. <u>The reduced form</u>. At the end of section 3 it was explained how an element z of Z leads to one of Z_o, called its reduced form. We shall denote it by rf(z), to be formally defined here for all $z \in Z'$:

(i) if $z = \overline{|z_1|,|z_2|}$ with $z_1 \in Z'$, $z_2 \in Z$ then

$$rf(z) = \overline{|rf(z_1)|,|rf(z_2)|},$$

(ii) if $z \in C$ then $rf(z)=z$,

(iii) if $z = \overline{|c|(|z_1|)|}$ with $c \in C$, $z_1 \in Z'$ then

$$rf(z) = \overline{|c|(|rf(z_1)|)|},$$

(iv) if $z = \sigma(n)$ for some $n \in \mathbb{N}$ then $rf(z) = z$,

(v) if $z = \overline{|\lambda|z_1|}$ with $z_1 \in Z$ then $rf(z) = \overline{|\lambda|rf(z_1)|}$,

(vi) if $z = \overline{|\sigma(\phi)|c|}$ with $\phi \in \Gamma$, $c \in C$ then $rf(z)=c$,

(vii) if $z = \overline{|\sigma(\phi)|c|(|x|)|}$ with $\phi \in \Gamma$, $c \in C$, $x \in Z'$ then

$$rf(z) = \overline{|c|(|rf(p_\phi(x))|)|},$$

where $p_\phi(x)$ is defined recursively by

$$p_\phi(\overline{|x|,|y|}) = \overline{|\phi|x|,|\phi|y|}, \quad p_\phi(y) = \overline{|\phi|y|} \qquad (x \in Z', y \in Z),$$

(viii) if $z = \overline{|\sigma(\phi)|\sigma(n)|}$ with $\phi \in \Gamma$, $n \in \mathbb{N}$ then $rf(z)=\sigma(\phi(n))$,

(ix) if $z = \overline{|\sigma(\phi)|\lambda|z_1|}$ with $\phi \in \Gamma$, $z_1 \in Z$ then $rf(z) = \overline{|\lambda|w|}$ with $w = rf(\overline{|\sigma(\phi^*)|z_1|})$, where ϕ^* is defined by $\phi^*(1)=1, \phi^*(k) = \phi(k-1)+1$ $(k=2,3,\ldots)$,

(x) if $z = \overline{|\sigma(\phi)|\sigma(\psi)|z_1|}$ with $\phi \in \Gamma$, $\psi \in \Gamma$, $z_1 \in Z$ then $rf(z) = rf(\overline{|\sigma(\phi\psi)|z_1|})$ (where, of course, $\phi\psi$ is defined by $(\phi\psi)(k)=\phi(\psi(k))$ for all $k \in \mathbb{N}$.

8. <u>Theorems on reduced forms</u>.

<u>Theorem 8.1</u>. For all $z \in Z'$ we have $rf(rf(z)) = rf(z)$.

<u>Theorem 8.2</u>. For all $z \in Z_o'$ we have $rf(z) = z$.

<u>Theorem 8.3</u>. If $\phi \in \Gamma$, $z \in Z$ then

$$rf(\overline{|\sigma(\phi)|z|}) = rf(\overline{|\sigma(\phi)|rf(z)|}).$$

Theorem 8.4. If ϕ_o is the identity ($\phi_o(n)=n$ for all n), and $z \in Z$, then $rf(\lceil \sigma(\phi_o) \mid z \rceil) = rf(z)$.

These theorems are easily proved by induction with respect to the length of z. At a certain point in the proof of Theorem 8.3 it plays a role that the operation of section 7 (ix) satisfies $(\phi\psi)^* = \phi^* \psi^*$.

9. Theorems on substitution.

Theoren 9.1. If Ω maps $\mathbb{N}$ into Z, and if $\phi \in \Gamma$, $z \in Z$ then

$$rf(subst(\Omega, rf(\lceil \sigma(\phi) \mid z \rceil))) = rf(subst(\Omega\phi, rf(z)).$$

Theorem 9.2. If Ω maps $\mathbb{N}$ into Z, and if $z \in Z'$ then

$$rf(subst(\Omega, z)) = rf(subst(\Omega_1, rf(z))),$$

where Ω_1 is defined by $\Omega_1(n) = rf(\Omega(n))$ for all n.

Theorem 9.3. If ϕ maps $\mathbb{N}$ into $\mathbb{N}$ and if $\Omega(n) = \sigma(\phi(n))$ for all n, then we have for all $z \in Z$

$$rf(subst(\Omega, z)) = rf(\lceil \sigma(\phi) \mid z \rceil).$$

Theorem 9.4. If Ω maps $\mathbb{N}$ into Z, if $z \in Z, \phi \in \Gamma$, and if Ω_1 is defined by $\Omega_1(n) = \lceil \sigma(\phi) \mid \Omega(n) \rceil$ (n=1,2,...) then

$$rf(subst(\Omega_1, z)) = rf(\lceil \sigma(\phi) \mid subst(\Omega, z) \rceil).$$

Theorem 9.5. If Ω, Σ, Λ are mappings of $\mathbb{N}$ into Z, such that

$$\Lambda(n) = subst(\Omega, \Sigma(n)) \qquad (n=1,2,\ldots)$$

then we have for all $z \in Z'$

$$rf(subst(\Omega, subst(\Sigma, z))) = rf(subst(\Lambda, z)).$$

These theorems provide a solid background to the conviction that rf(subst(Ω,z)) corresponds to what we usually mean by substitution. They are easily proved by recursion on the length of z. We omit the details.

10. Substitution in Z_o. Right now there is not enough experience to compare the value of the present system of substitution to other systems, in particular to the system of [1].

In order to facilitate the comparison, we present the definition of substitution of [1] in our present metalanguage. It operates on Z_o and Z'_o. If $z \in Z'_o$ and if Ω is a mapping of N into Z_o, the result of the substsitution will be denoted by $S(\Omega,z)$. The definition is by recursion:

(i) if $z = \boxed{\boxed{z_1}\,\boxed{,}\,\boxed{z_2}}$ with $z_1 \in Z'_o$ and $z_2 \in Z_o$ then

$$S(\Omega,z) = \boxed{\boxed{S(\Omega,z_1)}\,\boxed{,}\,\boxed{S(\Omega,z_2)}},$$

(ii) if $z \in C$ then $S(\Omega,z) = z$,

(iii) if $z = \boxed{\boxed{c}\,\boxed{(}\,\boxed{z_1}\,\boxed{)}}$ with $c \in C$, $z_1 \in Z'_o$ then

$$S(\Omega,z) = \boxed{\boxed{c}\,\boxed{(}\,\boxed{S(\Omega,z_1)}\,\boxed{)}},$$

(iv) if $z = \sigma(n)$ for some $n \in \mathbb{N}$ then $S(\Omega,z) = \Omega(n)$,

(v) if $z = \boxed{\boxed{\lambda}\,\boxed{z_1}}$ with $z_1 \in Z_o$ then

$$S(\Omega,z) = \boxed{\boxed{\lambda}\,\boxed{S(\Omega_1,z_1)}}$$

where Ω_1 is defined by its values $\Omega_1(1) = \sigma(1)$ and

$$\Omega_1(k) = S(\Gamma,\Omega(k-1)) \qquad (k=2,3,\ldots);$$

here Γ is the mapping defined by $\Gamma(k) = \sigma(k+1)$ for all k.

The fact that under (v) it is required to know the effect of S on expressions that are not subexpressions of z, makes recursion proofs a bit complex.

11. Algorithm for checking $rf(x) = rf(y)$. Let $x \in Z'$, $y \in Z'$. Quite often it is possible to answer the question whether $rf(x) = rf(y)$ without evaluating $rf(x)$ and $rf(y)$.

For every $z \in Z'$ there is a unique integer $k \geq 1$ such that z has the form $\boxed{\boxed{z_1}\,\boxed{,}\cdots\boxed{,}\,\boxed{z_k}}$ with $z_1 \in Z, \ldots, z_k \in Z$. Let us call k the string length of z and $z_1,\ldots,z_k$ the components of z. It is clear that z has the same string length as $rf(z)$. Hence x and y have different string length then certainly $rf(x) \neq rf(y)$.

Supposing x and y have the same string length k, we check whether $rf(x_1) = rf(y_1), \dots, rf(x_k) = rf(y_k)$. This means that we yet have to describe how we check $rf(x) = rf(y)$ if both x and y are in Z.

If $x = \lceil \sigma(\phi) \mid x_1 \rceil$, $y = \lceil \sigma(\phi) \mid y_1 \rceil$ with $\phi \in \Gamma$ we just replace the question y the one whether $rf(x_1) = rf(y_1)$.

If x still has the form $x = \lceil \sigma(\phi) \mid x_1 \rceil$ but if y does not have the form $\lceil \sigma(\phi) \mid y_1 \rceil$, we apply one of the reduction steps (vi) - (x) of section 7, and if the result is u, we ask whether $rf(u) = rf(y)$. We do a similar thing if this applies with x and y interchanged.

Finally, if neither x nor y have such a form, we say that $rf(x) \neq rf(y)$ unless we are in one of the following four cases:

(i) $x \in C$ and $y = x$,

(ii) $x = \lceil c \mid (\mid x_1 \mid) \rceil$, $y = \lceil c \mid (\mid y_1 \mid) \rceil$ with $c \in C$, $x_1 \in Z'$, $x_2 \in Z'$ and $rf(x_1) = rf(x_2)$.

(iii) $x = y = \sigma(n)$ for some $n \in \mathbb{N}$,

(iv) $x = \lceil \lambda \mid x_1 \rceil$, $y = \lceil \lambda \mid y_1 \rceil$ with $x_1 \in Z$, $x_2 \in Z$, $rf(x_1) = rf(x_2)$.

12. Remarks on uncombed forms. In examples, in particular in extensive ones, the comb notation is a nuisance, of course. In many circumstances we can omit the combs since it is clear where they have to be. This was discussed in [2].

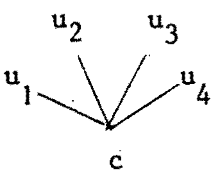
Instead of the $\sigma(n)$'s occurring in a formula we can simply write the corresponding n's provided that these n's are presented in standard decimal form. So for

$$\lceil g \mid (\mid \phi \mid \lambda \mid f \mid (\mid \omega \mid \sigma(3) \mid , \mid \sigma(5) \mid) \mid , \mid \psi \mid \lambda \mid f \mid (\mid \sigma(2) \mid , \mid \tau \mid \sigma(1) \mid) \mid) \rceil$$

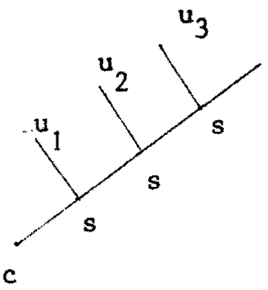
we can use the uncombed form (3.3).

We have to be careful if names for elements of C or Φ consist of more than one letter.

13. Remark on strings. Some of the notational effort of the previous sections went into the distinction between Z and Z', connected with the fact that we deal with n-ary expressions like $c(u_1, \dots, u_n)$. One of the disadvantages is, that recursion over the definition of Z is not so straight-forward as it might be. It is, of course, possible to eliminate this unpleasantness, removing all cases with $n > 2$. This can be done by creating a special constant s, and replacing, e.g.,



by



The cases n=0 and n=1 are left unaltered. The set of formulas is now defined as the minimal solution of

$$Q = C \cup \boxed{\boxed{C}\ (\ \boxed{Q}\ )} \cup \boxed{\boxed{s}\,(\ \boxed{Q}\ ,\ \boxed{Q}\ )} \cup N \cup \boxed{\lambda\,\boxed{Q}}\,.$$

We can consider the subset Q^* of formulas which do not start with an s and which never show ... s(s ... , This subset is closed under substitution: If in Q^* we substitute elements of Q^* we get a formula of Q^*.

References.

1. N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem.
Nederl.Akad.Wet.,Proc., Ser.A 75, (=Indag.Math. 34) 381-392 (1972).

2. N.G. de Bruijn. Notation for concatenation.
Technological University Eindhoven, Department of Mathematics.
Memorandum 1977-09. Issued August 1977.