Lambda Calculus Extended with Segments (vet)

Herman Balsters

Department of Mathematics and Computing Science

Eindhoven University of Technology

PO Box 513, 5600 MB, Eindhoven, The Netherlands.

1. Introduction

The aim of this paper is to give a description of a system of λ-calculus called $\lambda\sigma$. The system $\lambda\sigma$ is an extension of ordinary type free λ-calculus, the extension being that a new class of terms is incorporated in the system called segments. Segments are only parts of terms in the traditional sense and act as a sort of generalized contexts. Segments were originally introduced by N.G. de Bruijn (cf. de Bruijn[78,b]) in order to provide for certain abbreviational facilities in the mathematical language AUTOMATH. In this paper we shall treat the $\lambda\sigma$-system as an interesting extension of the λ-calculus in its own right and not pay attention to connections with AUTOMATH. For an introduction to the AUTOMATH project we refer to de Bruijn[80]. In the following sections of this paper we shall start from a simple system called λV and then gradually move on to the full $\lambda\sigma$-system.

2. The system λV

The system λV is the well-known type free λ-calculus as described in Barendregt[81], although there are some slight deviations in notation. Type free λ-calculus has formulas like

$$(xy) \tag{1}$$

$$\lambda x(xy) \tag{2}$$

$$\lambda x(\lambda y(xy)) \tag{3}$$

$$((\lambda x(\lambda y(xy)))z) \tag{4}$$

The corresponding formulas in λV are written as

$$\delta yx \tag{1'}$$

$$\lambda x\delta yx \tag{2'}$$

$$\lambda x\lambda y\delta yx \tag{3'}$$

$$\delta z\lambda x\lambda y\delta yx \tag{4'}$$

In λV functional abstraction is denoted by $\lambda x(...)$ (i.e. the function that assigns (...) to the variable x, where x may occur in (...)), and functional application is denoted by δAB (i.e. the function B applied to its argument A, where A and B are λV-terms). Note that in λV arguments are written in front of functions, this in contrast with ordinary type free λ-calculus where application of a function B to its argument A is usually written as (BA). The syntax of λV is very simple and is given below.

Definition

(1) λV-terms are words over the following alphabet

v1,v2,v3,...	variables
λ	abstractor
δ	applicator

(2) The set of λV-terms is the smallest set X satisfying

(i) $x \in X$,for every variable x

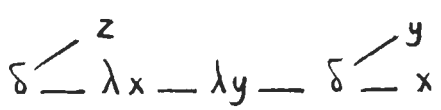
(ii) $A \in X \Rightarrow \lambda x A \in X$,for every variable x

(iii) $A, B \in X \Rightarrow \delta AB \in X$

As will be clear, λV-terms are written in prefix notation: each variable has arity 0, each abstractor λx has arity 1 and the applicator δ has arity 2. Each term can be represented as a rooted tree. As an example we consider the term

$$\delta z \lambda x \lambda y \delta y x \qquad (4')$$

which we write in tree-form as



(4'')

The correspondence between terms like (4') and trees like (4'') is one-to-one. It certainly helps to think of λV-terms as such trees, and in particular to see operations on terms as operations on their corresponding trees; especially when long terms are involved it is often useful to consider tree-representations of terms.

3. Substitution

In λ-calculus we have the fundamental notion of application. The application of a function B to an argument A is written as δAB. Apart from application we have the notion of functional abstraction. As said before, the intuitive meaning of $\lambda x(...)$ is ''the function that assigns (...) to x''. A term of the form $\delta A\lambda xB$ is called a redex. Substitution of A for the free occurrences of x in B is denoted by $\Sigma_x(A,B)$. The transition from $\delta A\lambda xB$ to $\Sigma_x(A,B)$ is called β-reduction. $\Sigma_x(A,B)$ is defined as usual by induction on the length of B. One remark must still be made, though. Unrestricted naive substitution leads to inconsistencies since free variables may get captured in the process. This problem is usually avoided by performing suitable renaming of bound variables (α-reduction). We would like to do without α-reduction altogether, namely by employing namefree notation. There are two reasons for employing this notation:

(1) avoidance of α-reduction,

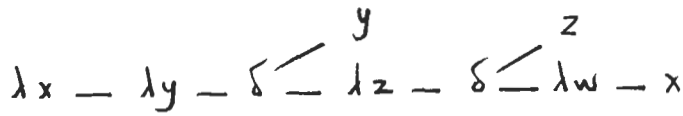
(2) precise notation for bindings of variables.

Apart from the fact that proofs involving α-reduction are notoriously tedious, we have our own intrinsic reasons to avoid α-reduction. Later on we shall introduce the full $\lambda\sigma$-system, an extension of λV. The main feature of $\lambda\sigma$ is the incorporation of a new class of terms called segments, desribed in section 5. Substitution of segments in name carrying form for their corresponding variables can give rise to a large number of complicated σ- reductions and even though name carrying notation could possibly be maintained, the namefree notation offers a much more concise description of substitution of segments and

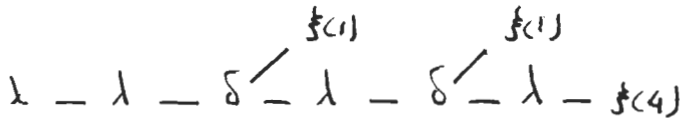
it is for this reason mainly that we have chosen for namefree representation of terms. We now proceed by explaining how namefree notation works.

4. Namefree notation

Namefree notation was first introduced by N.G. de Bruijn in his paper de Bruijn[72]. The idea is that we just write λ instead of $\lambda x, \lambda y, \lambda z, \ldots$ and every variable is replaced by a term of the form $\xi(n)$, where n is some positive integer. Each $\xi(n)$ is called a namefree variable and n is called its reference number. The reference number n of $\xi(n)$ determines the λ that binds a specific occurrence of $\xi(n)$ in some term . The procedure is as follows. If the namefree variable $\xi(n)$ occurs in some term t, we first form the tree-representation of t. We then descend from $\xi(n)$ towards the root of the tree and the n-th λ encountered is the λ which binds $\xi(n)$. As an example consider the following name carrying term in tree representation



The namefree equivalent of this term is



Remark If a reference number n is larger than the number of λ's lying

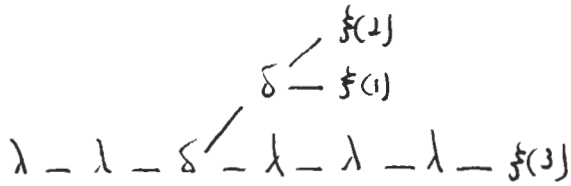
on the path from an occurrence of $\xi(n)$ to the root of the tree in which it occurs then we can interpret that occurrence as being free.

The use of namefree notation has certain consequences for substitution of $\lambda\xi$-terms (λV-terms written in namefree form), which we now shortly desribe. Substitution in a $\lambda\xi$-term t results in the replacement of free occurrences of a certain variable in t by some term u. We could also describe this situation in terms of trees by saying that certain end-points $\xi(n)$ of the tree-equivalent t^ of t have been replaced by some tree u^. Consider the following example of such a substitution in a $\lambda\xi$-tree.

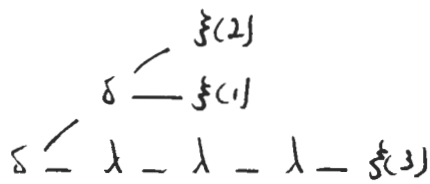
Let t be the $\lambda\xi$-term

$$\lambda\,\lambda\,\delta\,\delta\,\xi(2)\,\xi(1)\,\lambda\,\lambda\,\lambda\,\xi(3)$$

which has the following tree-representative t^

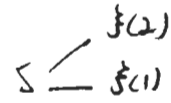


This tree contains a redex, namely

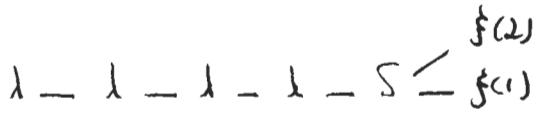


and we can therefore perform a β-reduction on t^. By β-reducing t^,

the end-point $\xi(3)$ is a candidate for substitution of the sub-tree



Should we, however, simply replace $\xi(3)$ by this sub-tree, as would have been the case if t^ had been written in name carrying form, then this would result in the following tree t'^



It is immediately clear that the variables $\xi(1)$ and $\xi(2)$ in t'^ refer to completely different λ's than in t^. This inconsistency is due to the fact that

(i) $\xi(1)$ and $\xi(2)$ are external references in t^ (i.e. references to λ's to the left of the subterm $\delta\xi(2)\xi(1))$
(ii) after replacement, the variables $\xi(1)$ and $\xi(2)$ in t'^ have two extra λ's on their left.

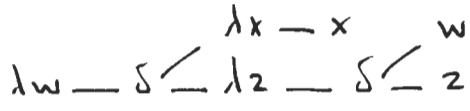
There is, however, a simple way to resolve this inconsistency: by raising the reference numbers 1 and 2 in $\xi(1)$ and $\xi(2)$ by 2 in t'^, these variables refer to the same λ's that they originally referred to in t^. This example demonstrates that certain measures have to be taken in order to ensure that external references remain intact when we substitute a $\lambda\xi$-term. One way of doing this is by employing so-called reference mappings (cf. de Bruijn[78,a]).We shall refrain from further discussion of this topic here, since this is intended as an informal paper and we wish to avoid technicalities.

5. Segments and abbreviations

In this section we shall describe a new class of terms called segments. We shall first stick to name carrying notation in order to simplify the discussion and to get the general flavor of what segments are and what we intend to do with them. Later on, in section 5.3, we shall describe how segments and segment variables can be written in namefree form.

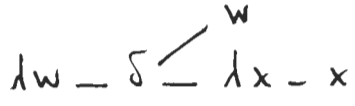
5.1 Variables as abbreviations

We may consider a variable as an abbreviation of a certain term if this variable can be replaced by that term by means of some suitable β-reduction. For example, consider the following term written in tree-form



(5)

By β-reducing (5) we obtain the term



(5')

i.e., a term in which the variable z has been replaced by the term λxx and the redex has vanished. If we have more occurrrences of the variable z, each bound by the λz of the redex, then each of these occcurrences serves as a kind of abbreviation of the term λxx.

In $\lambda\sigma$ there are , however, still quite different things that we wish to abbbreviate. One thing is a so-called δ-string like

$$\delta'_A \; \delta'_B \; \delta'_C \tag{6}$$

If it occurs more than once in a certain term, we may wish to abbreviate it. Yet (6) is not a term, in the sense of a λV-term, but only part of a term; it becomes a λV-term if we place an arbitrary λV-term behind it. Such parts of terms are called segments. Another example of a segment is a so-called λ-string like

$$\lambda x \_ \lambda y \_ \lambda z \tag{7}$$

In $\lambda\sigma$ much more general forms of segments are considered than δ-strings or λ-strings alone. In the following section we shall describe a general framework in which segments can be constructed.

5.2 Segment variables and substitution

Segments are terms with a kind of open end on the extreme right. From now on we shall use the symbol ω to indicate the open end on the extreme right. So

$$\delta'_A \; \delta'_B \; \delta'_C \; \omega$$

is a segment as well as

$$\lambda x \_ \lambda y \_ \lambda z \_ \omega$$

As said before, segments are not λ V-terms; a segment becomes a λ V-term if we replace the ω by an arbitrary λ V-term. According to this scheme the following formulas can also be considered as segments

$$\overset{A}{\sigma} - \lambda x - \lambda y - \omega$$

$$\lambda x - \overset{A}{\sigma} - \lambda y - \overset{B}{\sigma} - \omega$$

By replacing the ω in both of these formulas by some λ V-term we obtain a λV-term (provided, of course, that A and B are λV-terms). In $\lambda\sigma$ we will go even one step further by letting the ω's occur in other branches as well, like in

$$\lambda y - \overset{\lambda x - \omega}{\sigma} - \overset{y}{\sigma} - \omega$$

All of these occurrences of ω in the foregoing formulas act as a kind of "holes", which -once replaced by a λV-term - yield again a λV-term. All formulas having an ω on the extreme right are called segments in $\lambda\sigma$. Along with segments we also add to our system a new kind of variables for which segments can be substituted. These variables are represented by unary prefix symbols and are denoted, in name carrying form, by b,c,d,... . An example of a $\lambda\sigma$-term containing a segment and a segment variable is

$$\overset{\lambda x - \lambda y - \lambda z - \omega}{\sigma - \lambda b - b - x} \qquad (8)$$

This term is in redex form, where the segment variable b is bound by the λb of the redex. Performing a β-reduction on this segment results in

$$\lambda x \_ \lambda y \_ \lambda z \_ x \qquad (8')$$

i.e., the prefix symbol b is replaced by the segment $\lambda x \lambda y \lambda z$ (where the ω has been dropped). In $\lambda\sigma$, segment variables can serve as a means to abbreviate segments, just like variables in λV can serve as a means to abbreviate λV-terms. When using segment variables to abbreviate segments we must be careful, though. Consider for example the $\lambda\sigma$-term (8). The variable x in that term refers to the abstractor λx hidden inside the segment variable b, as seen in (8') where x gets bound by λx after β-reduction of (8). This is an intended feature which we always have to take into account in $\lambda\sigma$-calculus. If a segment variable b occurs in some $\lambda\sigma$-term t then after replacement of b by the segment s which that b abbreviates, it can happen, as most often will be the case, that certain variables occurring in t get captured by abstractors lying on the main branch of the tree representation of s. This is to say that each occurrence of a segment variable b in a $\lambda\sigma$-term t can contain abstractors -hidden inside b- which will capture certain variables in t after performing a β-reduction resulting in the replacement of b by the segment that b abbreviates in t.

In $\lambda\sigma$-calculus we also have the possibility of β-reduction inside segments. When performing a β-reduction inside a segment we are sometimes dealing with redices which, in the substitutional process in-

volved, have an effect on the ω on the extreme right of that segment. Consider, for example, the following segment

$$\lambda x - \overset{A}{\delta} - \lambda y - \lambda w - \lambda z - \omega \qquad (9)$$

By β-reducing the redex $\delta A \lambda y \lambda w \lambda z \omega$ occurring in (9) we are faced with evaluating $\Sigma y(A, \lambda w \overset{\omega}{\lambda z})$. We know how to "shift" the Σ y-operator past the two abstractors λw and λz, but then we arrive at the ω and have to decide how to evaluate $\Sigma y(A,\omega)$. We could simply define $\Sigma y(A,\omega)$ as ω, but then certain vital information would get lost; a situation which we now explain. Suppose that (9) occurs as a segment in some

term t and that (9) is referred to by some segment variable b occurring in t. Suppose also that there is an occurrence of the variable y in t which refers to the abstractor λy hidden inside b. By β-reducing (9) and defining $\Sigma y(A,\omega)$ as ω, this occurrence of y is no longer a candidate for substitution of the term A (which would have been the case prior to this β-reduction of (9)), simply because the abstractor λy has vanished. In order to avoid inconsistencies and to keep this candidate-role of substitution for such occurrences of variables y intact, we define such substitutions of a term A at an end point of a segment by

$$\Sigma y(A,\omega) = \overset{A}{\delta} - \lambda y - \omega$$

In this way it remains possible to refer to the λy of the original redex in (9), and occurrences of variables which referred indirectly to that lambda by means of a reference to a lambda hidden inside

some segment variable remain candidates for substitution of the term A. There is still one thing, however, which we have to take into consideration. The term A may contain free occurrences of w and z, and in order to ensure that these occurrences of w and z don't get captured, we may have to suitably rename the abstractors λw and λz to, say, $\lambda w'$ and $\lambda z'$. This means that β-reduction of (9) results in

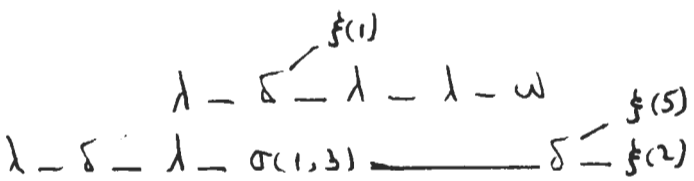
$$\lambda x \_ \lambda w' \_ \lambda z' \_ \delta \overset{A}{\diagup} \lambda y \_ \omega \qquad (9')$$

This still faces us with a problem, though. Suppose once more, that the segment (9) occurs in some $\lambda\sigma$-term t and that there also occur segment variables b in t which abbreviate (9) in t. Suppose furthermore that the variable z occurs in t to the right of an occurrence of b and that this occurrence of z refers to the abstractor λz lying on the main branch of (9). If we replace (9) in t by (9') then we would also have to rename the free occurrences of z to the right of b by z' in order to avoid inconsistencies. One can imagine, in general, that such situations can give rise to rather complicated α-reductions; complicated in the sense that this process has an accumulative effect and is not easy to describe formally. A precise formal definition can of course be given, but such a definition would be rather involved. There is a very concise way of describing substitutional effects in $\lambda\sigma$-calculus, namely by employing namefree notation for segments and segment variables. This notation is described in the following section.

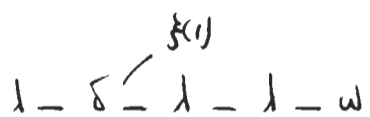
5.3 Namefree notation for segments and segment variables

In this section we shall describe the full $\lambda\sigma$-system: the namefree

system $\lambda\xi$ extended with segments and segment variables. What we shall do is the following. Segment variables are written in namefree form as $\sigma(n,m)$, where n denotes the reference number of σ (which, like in $\xi(n)$, determines the λ that some specific occurrence of $\sigma(n,m)$ refers to) and m ($m \geq 0$) denotes the number of λ's lying on the main branch of the tree-representation of the segment that $\sigma(n,m)$ intends to abbreviate (the number m is also called the weight of $\sigma(n,m)$). As an example of a term in namefree notation containing a segment and a segment variable consider the following term written in tree form

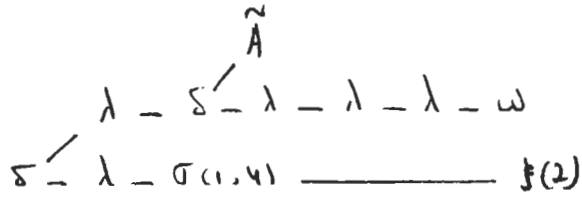


In this term we see that $\sigma(1,3)$ abbreviates a segment with three λ's lying on the main branch of its tree; so when determining the λ that $\xi(5)$ refers to, we descend from $\xi(5)$ towards the root of the tree, subtract 3 from 5, subsequently subtract 1 and see that $\xi(5)$ refers to the first λ (from the left) of the tree. The variable $\xi(2)$ refers to the second λ (from the right) hidden inside $\sigma(1,3)$; $\xi(2)$ is thus bound by the second λ (from the right) of the segment



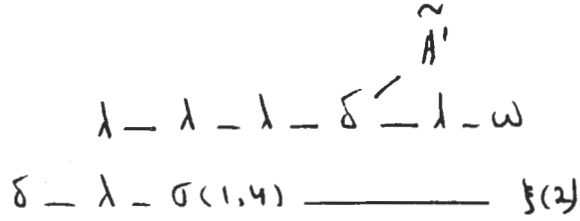
Namefree notation for segments and segment variables offers a very precise desription of bindings (especially hidden bindings via segment variables) in $\lambda\sigma$-terms. Namefree notation also offers an alternative for performing complicated α-reductions due to the effect of certain β-reductions inside segments as described in section 5.2, which we now

shortly describe. We recall that the performance of certain β-reductions inside segments prior to substitution of those segments for their respective segment variables gave rise to problems concerning references to the λ's of the redices contracted. In λr we can resolve such inconsistencies by adding an extra parameter, called a segment mapping, to an ω. In order to illustrate how this is done consider the following term in which the segment (9) occurs, written in namefree form



(10)

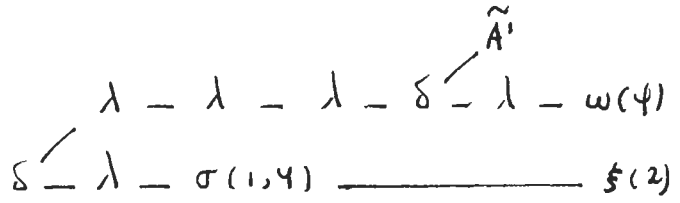
where A~ denotes the namefree equivalent of A. This term seems to β-reduce to



(10')

where A'~ is obtained from A~ by raising all external references in A~ (those reference numbers in A~ which refer to some λ to the left of A~ in (10)) by the number 2, thus ensuring that occurrences of variables in A'~ refer to the same λ's that they originally referred to in (10). But now we arrive at the same unwanted situation as in (9'): just like a free occurrence of a variable z no longer refers to its original abstractor λz (λz has been replaced by λz'), the variable ξ(2) also no longer refers to its original λ in (10'). We shall resolve such inconsistencies by adding so-called segment mappings (or segmaps for short) to end-points ω and write ω(ψ), where ψ is a segmap. A segmap is a

permutation of some interval [1..n] of $\mathbb{N}$, and tells us how to reallocate references to λ's hidden inside segment variables. In the case of our example we replace the ω in (10') by $\omega(\psi)$, thus obtaining



(10'')

where ψ is a permutation of the interval [1..4] defined by

$\psi(1)=2$, $\psi(2)=3$, $\psi(3)=1$ and $\psi(4)=4$.

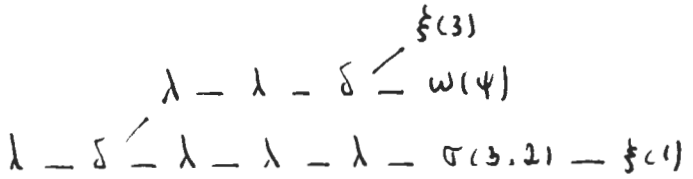
This segmap is to be interpreted as follows: references to the first λ (from the right) are now reallocated to the second λ (from the right) lying on the main branch of the segment

;

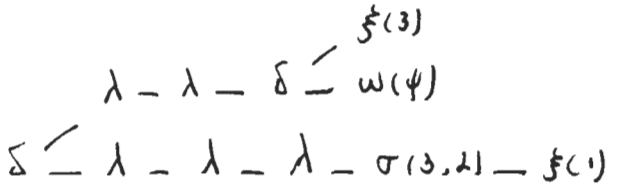
furthermore: references to the second λ (again, from the right) are reallocated to the third, references to the third λ are reallocated to the first (the most vital reallocation) and references to the fourth λ remain unaltered. By adding this segmap ψ we have an instrument for reallocating references in such a way that all occurrences of variain (10'') refer to the same λ's they originally referred to in (10). A further discussion of substitution of segments involving segmaps is given in the following section.

5.4 Substitution of segments for segment variables in namefree form

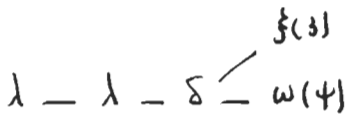
In this section we shall shortly describe substitution for segment variables written in namefree form. We shall give this description in an informal manner in terms of trees. The tree representation of a segment has an $\omega(\psi)$ -where ψ is some segmap on the extreme right of its main branch. When we substitute a segment we remove the $\omega(\psi)$ and put the remaining tree-fragment in the place of some occurrence of a segment variable in a $\lambda\sigma$-tree. Segment variables occur in $\lambda\sigma$-trees as unary nodes and substitution of segments for segment variables thus gives rise to replacements at unary nodes inside a $\lambda\sigma$-tree (which differs completely from $\lambda\xi$-substitutions, where we could only perform replacements at end-nodes of trees). When such a substitution is performed, we again -as in the case of $\lambda\xi$-substitutions- have to be careful and update external references in segments in order to ensure that these references remain intact after substitution. But not only do we have to update external references when we substitute a segment for a corresponding segment variable, we also have to take into account the effect of the segmap ψ attached to the end-point of the segment involved, since such a segmap reallocates references to λ's lying on the main branch of the segment which we want to substitute. We now give an example to demonstrate both of these features. Consider the following example of a $\lambda\sigma$-tree containing a segment and a segment variable



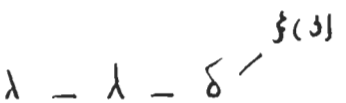
where ψ is the permutation of [1..2] defined by $\psi(1)=2$ and $\psi(2)=1$. This tree, which we refer to as t^, contains a redex, namely



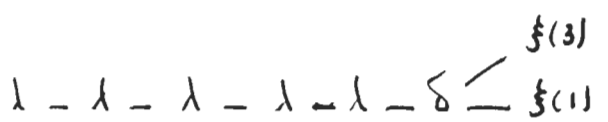
and we can therefore perform a β-reduction on t^. By β-reducing t^, the unary node $\sigma(3,2)$ is a candidate for substitution of the subtree



Should we simply replace $\sigma(3,2)$ by the tree-fragment

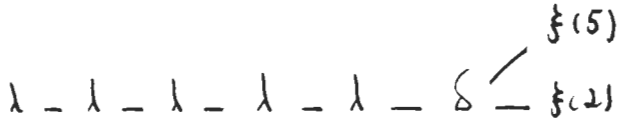


then this would result in the following tree t''^



It is immediately clear that the variables $\xi(1)$ and $\xi(3)$ refer to λ's different from those that they originally referred to in t^. The variable $\xi(3)$ is an external reference in t^ and, as in the case of $\lambda\xi$-substitutions, has to be suitably updated whenever the segment in which $\xi(3)$ occurs is substituted for some segment variable. The variable $\xi(1)$ in t^ refers to one of the two λ's hidden inside $\sigma(3,2)$; it seems to refer to the first λ (from the right) lying on the main branch of the segment involved, but the segmap ψ reallocates this re-

ference to the second λ (from the right). This means that correct β-reduction of t^ would result in the following tree t"^



The employment of namefree notation and segmaps makes it possible to to give a formal definition of substitution of segments for segment variables in a very concise way. We shall not give such a definition here, we just state this as a fact. Readers interested in technical matters concerning $\lambda\sigma$-calculus are referred to the author's forthcoming thesis, which covers language-theoretical aspects of the $\lambda\sigma$-system. We note that in previous examples describing how substitution of segments for segment variables can take place, we have restricted ourselves to rather simple situations. Our formal treatment of such substitutions, however, will take much more involved situations into account. Our formal definition of substitution will take into consideration certain accumalative effects which can occur when segments contain references to other segments, or even λ's which bind segment variables.

6. Conclusions and results

As stated earlier, the language theory of $\lambda\sigma$-calculus is the topic of the author's forthcoming thesis. Both an untyped and a typed version of the $\lambda\sigma$-system are treated in this thesis. The types in the typed version of $\lambda\sigma$ (called $\lambda_T\sigma$) are an extension of the types in Church's well-known Theory of simple types, the extension being that simple types are constructed for segments and segment variables. The follow-

ing important language theoretical results have been established:

(1) the Church-Rosser property for $\lambda\sigma$ (and $\lambda_T\sigma$),
(2) the normalization property for $\lambda_T\sigma$.

With these two results, soundness and decidability of the $\lambda\sigma$-system are guaranteed.

The $\lambda\sigma$-system was originally devised for theoretical research concerning the foundation of the AUTOMATH verifying programs which are used for the checking of correctness of mathematical texts coded in AUTOMATH. One application of the $\lambda\sigma$-system is that it provides a basis for a general purpose language in which AUTOMATH-texts can be coded in such a way that the verification of correctness can be done more easily and systematically than before. But since the $\lambda\sigma$-system provides for a large number of new abbreviational facilities in λ-calculus, more applications than those strictly connected with AUTOMATH seem possible.

7. References

Barendregt[81]: Barendregt, H.P. The Lambda Calculus:Its Syntax and Semantics. North Holland, 1981.

de Bruijn[72]: de Bruijn, N.G. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indag. Math. 34, 1972, pp. 381-392.

de Bruijn[78,a]: de Bruijn, N.G. Lambda calculus with namefree formulas involving symbols that represent reference transforming mappings.

Indig. Math. 40, 1978, pp. 348-356.

de Bruijn[78,b]: de Bruijn, N.G. A Namefree Lambda Calculus with Facilities for Internal Definition of Expressions and Segments. The University of Technology, Dept. of Mathematics, Eindhoven, 1978, TH Report 78-WSK-03.

de Bruijn[80]: de Bruijn, N.G. A survey of the project AUTOMATH. Seldin & Hindley[80], pp. 579-607.

Church[40]: Church, A. A formulation of the simple theory of types. J. Symbolic Logic 5, 1940, pp.56-68.

Seldin & Hindley[80]: Seldin, J.P. and Hindley, J.R. To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, edited by J.P. Seldin and J.R. Hindley. Academic Press, 1980.