COMBINATORS AND LAMBDA-CALCULUS,
A SHORT OUTLINE.

J. Roger Hindley,
Mathematics Department,
University College, Swansea SA2 8PP,
Wales, U. K.

Le Val d'Azul, 6-5-1985.

ABSTRACT
This article introduces the basic ideas of combinatory logic and lambda-calculus, to serve as background for the other papers in this volume. Typed and untyped systems are covered.

0. INTRODUCTION

Combinatory logic and λ-calculus can both be viewed as programming languages, though they were originally invented in the 1920's during philosophical investigations into the foundations of mathematics.

Their principal feature is that they are higher-order, that is, they give a systematic notation for operators whose input and output values may be other operators. Also, they are not based on the concept of set like many mathematical languages are, but on that of function, or operator.

The λ-calculus (or really λ-calculi, because there are many) was invented by Alonzo Church; see his 1941 for an overview. Combinatory logic was invented by Moses Schönfinkel (see his 1924) and re-invented and developed by Haskell Curry. For a brief history, see Curry et al. 1958 section 581.

It may seem surprising that two 60-year-old languages have roles to play in today's programming-theory, but there is a good reason for this. In trying to automate considerable parts of mathematical and logical thought, programming-theorists are now meeting the same questions that have puzzled logicians for the past 80 years or so, particularly in the various schools of constructive mathematics. So, unless these logicians have gone
astray completely, their work will at the very least be a useful source of ideas.

This paper is a very quick introduction to the main ideas, in their simplest form. All details and minor variations will be omitted.

Further information is in the basic introductory book Hindley and Seldin 1985, and references such as '(HS 6.34)' will refer to paragraphs in that book.

Even more information is in Barendregt 1984, and, for older material, Curry et al. 1958, 1972.

1. TYPE-FREE $\lambda$-CALCULUS

1.1 PRELIMINARY NOTE. Functions of two or more arguments will not be considered here. The reason is just to keep the paper short. But it is worth noting that, in any language with a notation for higher-order operators, a 2-argument operator $h$ can always be 'represented' by a one-argument operator $h^*$, defined as follows. First, for each $x$, define an operator $h_x$ thus:

$$h_x(y) = h(x,y);$$

Then define $h^*$ by

$$h^*(x) = h_x.$$

Then, for all $x, y$, we have

$$(h^*(x))(y) = h_x(y) = h(x,y).$$

The act of replacing $h$ by $h^*$ is often called Currying, after Haskell Curry. But in fact Curry refers to Schönfinkel 1924 for it, and it also occurs in several 19th-century publications, I believe.

1.2 LAMBDA-NOTATION. An arithmetical expression such as $'x+2y'$ can be viewed as defining either a function of $x$, with $y$ held constant, or a function of $y$, with $x$ held constant. Church called these functions, respectively, $\lambda x.x+2y$, $\lambda y.x+2y$.

We have

$$(\lambda x.x+2y)(a) = a+2y,$$

$$(\lambda y.x+2y)(a) = x+2a.$$  

If $h(x,y) = x+2y$, then the $\lambda$-notation for our previous $h^*$ is

$$h^* = \lambda x. (\lambda y.x+2y).$$

For all $a, b$, we have
The \( \lambda \)-notation allows us to distinguish very clearly between \( h^* \) and its 'reverse' operator; in this notation, they are \( \lambda x. (\lambda y. x+2y) \) and \( \lambda y. (\lambda x. x+2y) \).

This informal \( \lambda \)-notation will be formalized in the following definitions. For simplicity, I shall state them without extra constants such as '+' and '2', though \( \lambda \)-systems used in practice always contain some.

1.3 DEFINITION (HS 1.1): \( \lambda \)-terms. Assume that we have an infinite sequence of variables \( x, y, z, x_1, y_1, z_1, x_2, \) \( y_2, z_2, \ldots \) (to denote arbitrary operators). Construct \( \lambda \)-terms as follows:

(a) each variable is a \( \lambda \)-term;
(b) from any \( \lambda \)-terms \( X, Y \), construct \( (XY) \) (to denote the application of operator \( X \) to input \( Y \));
(c) from any variable \( x \) and \( \lambda \)-term \( M \), construct \( (\lambda x. M) \) (to denote the function of \( x \) that \( M \) defines).

We call \( (XY) \) an application, and \( (\lambda x. M) \) an abstraction.

1.4 NOTATION. Letters \( U, V, W, X, Y, Z, M, N, P \) will denote arbitrary \( \lambda \)-terms. The identity
\[
x = Y
\]
will mean that \( X \) is the same term as \( Y \) (syntactic identity).

Parentheses and repeated \( \lambda \)'s will often be omitted; for example
\[
X Y Z M = (((XY)Z)M),
\]
\[
\lambda x y z. M = (\lambda x. (\lambda y. (\lambda z. M))).
\]
The application of \( X \) to \( Y \) is often called \( X(Y) \) in mathematics; the reason that it is called \( (XY) \) here is merely historical.

1.5 DEFINITION (HS 1.10): free and bound variables. Any occurrence of a variable \( x \) in a term \( \lambda x. M \) is called bound. Any non-bound occurrence in a term is called free. A term with no free variables is called a closed term or a combinator. The set of all variables that have free occurrences in \( X \) is called \( \text{FV}(X) \).

1.6 EXAMPLE. Let
\[
X = \lambda y. y x (\lambda x. y y x).
\]
Then all 3 \( y \)'s are bound in \( X \), the leftmost \( x \) is free in \( X \),
the next 2 x's are bound in X, and the v is free in X.
Also
\[ \text{FV}(X) = \{x, v\} \].

1.7 DEFINITION (HS 1.11): substitution. \([N/x]M\) is the result of substituting \(N\) for each free occurrence of \(x\) in \(M\), and changing any \(\lambda y\)'s in \(M\) to prevent variables free in \(N\) from becoming bound in \([N/x]M\). In more detail:

(a) \([N/x]x \equiv N; \]
(b) \([N/x]y \equiv y \quad \text{if} \ y \not\equiv x; \]
(c) \([N/x](PQ) \equiv ([N/x]P) [N/x]Q; \]
(d) \([N/x](\lambda x.P) \equiv \lambda x.P; \]
(e) \([N/x](\lambda y.P) \equiv \lambda y.[N/x]P \quad \text{if} \ y \not\in \text{FV}(N) \text{ or } x \not\in \text{FV}(P); \]
(f) \([N/x](\lambda y.P) \equiv \lambda z.[N/x](z/y)P \quad \text{if} \ y \in \text{FV}(N) \text{ and } x \in \text{FV}(P), \]
where \(z\) is the first variable not in \(\text{FV}(N)\).

1.8 EXAMPLES.

\[
[v/x](\lambda y.yx) \equiv \lambda y.[v/x](yx) \quad \text{by (e)}
\]
\[
[v/x](\lambda v.vx) \equiv \lambda z.[v/x](zx) \quad \text{by (f)}
\]
\[
[v/x](\lambda v.vx) \equiv \lambda z.zv \quad \text{by (a)}.
\]

If (f) were omitted from Definition 1.7, then we would have the undesirable fact that, although \(\lambda v.x\) and \(\lambda y.x\) both denote the same operator (the constant-operator whose output is always \(x\)), they would come to denote different operators when \(v\) was substituted for \(x\):

\[
[v/x](\lambda y.x) \equiv \lambda y.v, \quad [v/x](\lambda v.x) \equiv \lambda v.v.
\]

1.9 NOTE: Changing bound variables. Both the terms \(\lambda x.x\) and \(\lambda v.v\) denote the identity-operator. In general, terms that differ only by changing bound variables have the same meaning.

The process of changing bound variables is defined formally as follows.

1.10 DEFINITION (HS 1.16): \(\alpha\)-conversion. If \(y \notin \text{FV}(P)\), we say

(a) \(\lambda x.P \equiv_\alpha \lambda y.(y/x)P.\)

If \(X\) changes to \(Y\) by a finite (perhaps empty) series of replacements of form (a), we say

\(X \equiv_\alpha Y,\)

or \(X\) is congruent to \(Y\), or \(X\) \(\alpha\)-converts to \(Y\). The relation \(\equiv_\alpha\) can be proved symmetric (HS 1.17(c)).
1.11 DEFINITION (HS 1.22): $\beta$-contraction, $\beta$-reduction.
A term of form $(\lambda x . M) N$
represents an operator applied to an input; it is called a $\beta$-redex. If it occurs in a term $X$, and we replace one occurrence of it by
$\frac{\text{[N/x]}}{\text{M}}$
then we say that we have contracted that occurrence of it. A finite (perhaps empty) or infinite series of contractions and changes of bound variables is called a $\beta$-reduction. If it is finite, and changes $X$ to $Y$, we say that $X$ $\beta$-reduces to $Y$, or
$X \rightarrow_{\beta} Y$.

1.12 EXAMPLE. Here are some reductions; the redex contracted at each step is underlined. In (c), the reduction is infinite, though the term never changes.

(a) $(\lambda x . ((\lambda y . x y) u)) (\lambda v . v)$ $\rightarrow_{\beta} (\lambda y . (\lambda v . v) y) u$
    $\rightarrow_{\beta} (\lambda y . y) u$
    $\rightarrow_{\beta} u$.

(b) $(\lambda x . ((\lambda y . x y) u)) (\lambda v . v)$ $\rightarrow_{\beta} (\lambda x . x u) (\lambda v . v)$
    $\rightarrow_{\beta} (\lambda v . v) u$
    $\rightarrow_{\beta} u$.

(c) $(\lambda x . x x) (\lambda x . x x)$ $\rightarrow_{\beta} (\lambda x . x x) (\lambda x . x x)$
    $\rightarrow_{\beta} ...$

1.13 DEFINITION (HS 1.24): $\beta$-normal form. A term $N$
containing no $\beta$-redexes is called a $\beta$-normal form, or $\beta$-nf. If $M$ $\beta$-reduces to $N$, then $N$ is called the $\beta$-normal form of $M$.

Not every term has a $\beta$-normal form; for example the term in Example 1.12(c) has none. Examples 1.12(a) and (b) raise the question of whether the nf is always unique. The next theorem shows that it is.

1.14 CHURCH-ROSSER THEOREM FOR $\beta$-REDUCTION (HS 1.29).
If $U \rightarrow_{\beta} X$ and $U \rightarrow_{\beta} Y$ (see Figure 1), then there exists $Z$ such that
$X \rightarrow_{\beta} Z$, $Y \rightarrow_{\beta} Z$. 
Proof. HS, Appendix I.

Figure 1.

1.14.1 COROLLARY (HS 1.29.1). If a term has a \( \beta \)-nf, then that nf is unique, modulo \( \equiv_\alpha \).

1.15 DEFINITION (HS 1.32): \( \beta \)-equality or \( \beta \)-conversion.
When we can change \( P \to Q \) by a finite series of \( \beta \)-reductions and reversed \( \beta \)-reductions (called \( \beta \)-expansions), we say that \( X \beta \)-converts to \( Y \) or \( X \) is \( \beta \)-equal to \( Y \), or
\[ X =_\beta Y. \]

1.16 CHURCH-ROSSER THEOREM FOR \( \beta \)-EQUALITY (HS 1.35).
If \( X =_\beta Y \), then there exists \( Z \) such that \( X \beta \equiv Z \) and \( Y \beta \equiv Z \).

Proof. By Theorem 1.14; see Figure 2.

Figure 2.

1.16.1 COROLLARY (HS 1.35.3). If \( X \) and \( Y \) are \( \beta \)-nf's, then
\[ X =_\beta Y \iff X \equiv_\alpha Y. \]

1.16.2 COROLLARY. The theory of \( =_\beta \) is consistent, in the sense that not all terms can be proved equal.

Proof. \( \lambda x.x \) and \( \lambda x y.y x \) are distinct \( \beta \)-nf's.
I.17 NOTE. The following combinators have proved useful enough to receive individual names (and some have several names):

1. Identity combinator: \( I \equiv \lambda x.x \)
2. Composition combinator: \( B \equiv \lambda xyz.x(yz) \)
3. Commutator: \( C \equiv \lambda xyz.xzy \)
4. Substitution & composition: \( S \equiv \lambda xyz.xz(yz) \)

Truth-values, or selectors:

1. 1st selector, or truth: \( K \equiv \lambda xy.x \)
2. 2nd selector, or falsehood: \( \tilde{O} \equiv \lambda xy.y \)

Numerals: \( n \) is often represented by an \( n \)-fold composition term called \( \tilde{n} \) or \( 2^n \):

- \( 0 = \lambda xy.y \)
- \( 1 = \lambda xy.x \)
- \( n = \lambda xy.x(...(xy)...)(n \text{'s}). \)

Successor-combinator:

\( \tilde{S} \equiv \lambda xy.x(uxy) \)

Ordered-pair combinator, or test for zero (HS 4.7):

\( D \equiv \lambda xyz.z(Ky)x \)

Ordered-combinator may be called "if \( Z = 0 \) then \( X \) else \( Y \)."

Primitive-recursion combinator (HS 4.10):

\( R \equiv \lambda xy.u((0y)(\tilde{O}x))I \)

where \( Q \equiv \lambda yv.0(\tilde{O}(v0))y(v0)(v0) \).

(Note that, contrary to a popular legend, the fixed-point combinator is not needed to define \( R \). (Though it could have been used if we had wished, HS 4.11.)

Fixed-point combinator (HS 3.4):

\( Y \equiv \lambda x.((\lambda y.x(yy))(\lambda y.x(yy))) \)

\( YF = \beta F(YF) \)

(For all \( F \), if \( X \equiv YF \), then \( FX = \beta X \).)

1.18 DEFINITION (HS 4.4). A \( k \)-argument partial function \( \phi \) of natural numbers is \( \lambda \)-definable by a term \( X \) when

\( X\tilde{n}_1...\tilde{n}_k \) \( \beta \) \( \phi(n_1,...,n_k) \) if \( \phi(n_1,...,n_k) \) exists

\( X\tilde{n}_1...\tilde{n}_k \) has no \( \beta \)-nf otherwise.
1.19 THEOREM (HS 4.18). A partial function of natural numbers is \( \lambda \)-definable if and only if it is partial recursive.

1.19.1 COROLLARY (HS 5.6.3). The relations \( \beta \) and \( \beta' \) are not recursively decidable.

2. TYPE-FREE COMBINATORY LOGIC

Combinatory logic is a way of avoiding the complications with bound variables that spoil the \( \lambda \)-calculus. In this chapter, \( \lambda \)-abstraction will be defined in terms of other primitives called basic combinators, not taken as primitive itself. These other primitives are slightly less intuitive than \( \lambda \), but are technically much simpler, as we shall see.

2.1 DEFINITION (HS 2.1): \( \text{CL-terms} \). Start with an infinite list of \textbf{variables}, the same as in \( \lambda \)-calculus, plus two extra symbols \( K \) and \( S \), called \textbf{basic combinators}. From these, construct \( \text{CL-terms} \) thus:

- (a) \( K \), \( S \), and the variables, are \( \text{CL-terms} \);
- (b) from any \( \text{CL-terms} \) \( X \), \( Y \), construct \( (XY) \).

2.2 NOTATION. The same as in Chapter 1, except for the following simplifications:

- \( \text{FV}(X) \) now means the set of all variables occurring in \( X \), since variables cannot now be bound.
- A \textbf{closed term} or \textbf{combinator} is a term whose only atoms are \( K \) and \( S \).
- Substitution, \([N/x]M\), now means simply replacing each \( x \) in \( M \) by \( N \).

2.3 EXAMPLES OF \( \text{CL-TERMS} \):

\[
\begin{align*}
I &= SKK \quad \text{[this is really \((SK)K\), of course]}, \\
B &= S(KS)K, \\
C &= S(BBS)(KK), \\
\text{\texttt{e}} &= K1, \\
\text{\texttt{f}} &= SD, \\
\text{\texttt{n}} &= \text{\texttt{e}}(\text{\texttt{f}}(...(\text{\texttt{f}})...)) \quad (n = \text{\texttt{f}}'s).
\end{align*}
\]
2.4 Definition (HS 2.7): weak contraction, reduction.
A weak redex in a term $U$ is any occurrence of a term of form $KXY$, $SXYZ$.
A weak contraction is the act of replacing a weak redex by, respectively,
$X$, $XZ(YZ)$.
A series of weak contractions is called a weak reduction. If it is finite, and changes $U$ to $V$, we say $U$ weakly reduces to $V$, or $U \xrightarrow{w} V$.

2.5 Example. $I$ acts as an identity-operator, because
$IX = SKKK$ by contracting $SKKK$
$\xrightarrow{w} X$ by contracting $KK(KX)$.

2.6 Example. $B$ acts as a composition-operator, because
$BFGX = S(KS)KFGX$
$\xrightarrow{w} S(KS)(KF)(GX)$ by contracting $S(KS)KF$
$\xrightarrow{w} S(KF)(GX)$ by contracting $KSF$
$\xrightarrow{w} KFX(GX)$ by contracting $S(KF)GX$
$\xrightarrow{w} F(GX)$ by contracting $KFX$.

2.7 Exercise. Prove that
(a) $CFXY \xrightarrow{w} FYX$,
(b) $\overline{F}FX \xrightarrow{w} F(F(...(FX)...))$ (n F's).

2.8 Definition (HS 2.14): abstraction. For each $x$ and each term $M$, we construct a new term called $\lambda^x.M$, built up from $K$'s and $S$'s and parts of $M$, by induction on the number of symbols in $M$, as follows.
(a) $\lambda^x.U = KU$ if $x \not\in FV(U)$,
(b) $\lambda^x.x = I$,
(c) $\lambda^x.Ux = U$ if $x \not\in FV(U)$,
(f) $\lambda^x.UV = S(\lambda^x.U)(\lambda^x.V)$ if (a)-(c) do not apply.

2.9 Example.
$\lambda^x.xy = S(\lambda^x.x)(\lambda^x.y)$ by (f)
$= SI(Ky)$ by (b), (a).
2.10 DEFINITION (HS 2.17): multiple abstraction. Define
\[ \lambda^*x_1\ldots x_n.M \equiv \lambda^*x_1.(\lambda^*x_2(\ldots(\lambda^*x_n.M)\ldots)). \]

2.11 EXAMPLE. \[ \lambda^*y.x.xy \equiv \lambda^*y.(\lambda^*x.xy) \]
\[ = \lambda^*y.(SI(Ky)) \]
\[ = S(\lambda^*y.SI)(\lambda^*y.Ky) \]
\[ = S(K(SI))K \]
by 2.9
by 2.8(f)
by 2.8(a), (c).

2.12 THEOREM (HS 2.15). For all \( x, M, N \),
\[ (\lambda^*x.M)N \Downarrow_w [N/x]M. \]

Proof. Induction on the number of symbols in \( M \). Note how the contractions for \( K \) and \( S \) correspond exactly to Cases (a) and (f) below.
(a) If \( x \notin \text{FV}(M) \):
\[ (\lambda^*x.M)N \equiv \lambda^*MN \]
\[ \Downarrow_w M \equiv [N/x]M. \]
(b) If \( M \equiv x \):
\[ (\lambda^*x.x)N \equiv IN \]
\[ \Downarrow_w N \equiv [N/x]x. \]
(c) If \( M \equiv Ux \) and \( x \notin \text{FV}(U) \):
\[ (\lambda^*x.Ux)N \equiv UN \]
\[ \Downarrow_w [N/x](Ux). \]
(f) If \( M \equiv UV \) and (a) - (c) do not apply:
\[ (\lambda^*x.UV)N \equiv S(\lambda^*x.U)(\lambda^*x.V)N \]
\[ \Downarrow_w ([N/x]U)([N/x]V) \text{ by induction hypoth.} \]
\[ = [N/x](UV). \]

2.13 DEFINITION (HS 2.8) A weak normal form, or wnf, is any CL-term that contains no weak redexes. The wnf of \( X \) is any wnf, \( N \), such that \( X \Downarrow_w N. \)

Not every CL-term has a weak normal form. For example,
\[ \text{SII}(\text{SII}) \]
has none. But a term cannot have two wnf's, as the following theorem shows.

2.14 CHURCH-ROSSER THEOREM FOR WEAK REDUCTION (HS 2.13).
If \( U \Downarrow_w X \) and \( U \Downarrow_w Y \), then there exists \( Z \) such that
\[ X \Downarrow_w Z, \quad Y \Downarrow_w Z. \]

Proof. HS, Appendix 1.
2.11.1 COROLLARY (HS 2.13.1). If $X$ has a wnf, then it is unique.

2.15 DEFINITION (HS 2.22): \textit{weak equality}. When we can change $X$ to $Y$ by a finite series of weak contractions and reversed weak contractions, we say $X$ is weakly equal to $Y$, or $X \equiv_w Y$.

2.16 CHURCH-ROSSER THEOREM FOR WEAK EQUALITY (HS 2.24). If $X \equiv_w Y$, then there exists $Z$ such that $X \equiv_w Z$ and $Y \equiv_w Z$.

2.16.1 COROLLARY (HS 2.24.3). If $X$, $Y$ are distinct wnf's, then $X \not\equiv_w Y$. Hence the theory of $\equiv_w$ is consistent.

2.17 NOTE. Because abstraction has been defined in combinatory logic, all the $\lambda$-terms in Note 1.17 can be defined in CL too. Also, all the partial recursive functions are combinatorially definable (i.e. definable by a CL-term $X$ in a way similar to Definition 1.18). Hence the relations $\equiv_w$ and $\not\equiv_w$ are not recursively decidable.

2.18 WARNING. CL and $\lambda$ are not perfectly parallel, however. There is one important property that $\lambda$ has but CL has not, namely the property that Curry called (E):

\[(E) \quad X = Y \implies \lambda x. X = \lambda x. Y.\]

For $\lambda$-equality, (E) is true because any contractions or expansions in $X$ can also be made in $\lambda x. X$. But for CL-terms and weak equality, the analogue of (E) is

\[(E)_{\text{CL}} \quad X \equiv_w Y \implies \lambda^* x. X \equiv_w \lambda^* x. Y,\]

and this fails. For example, take

$X \equiv Kxx$, $Y \equiv x$.

3. TYPED $\lambda$-CALCULUS

The motivation for typed $\lambda$-calculus is the view that a function is not well-defined until its domain and range have been defined, and that a term $X$ that represents a function should display its domain and range as part of its structure.

We begin by defining 'types', which are expressions intended to denote sets.
3.1 DEFINITION (HS13.1): *TYPES*. Select some symbols and call them *atomic types*; for example, \( 0 \) to denote the set of all natural numbers, and \( 1 \) to denote the set \( \{ \text{true}, \text{false} \} \). From any types \( \alpha, \beta \), construct a new type

\( (\alpha \rightarrow \beta) \)

to denote a set of functions from \( \alpha \) into \( \beta \).

Example. \( (0 \rightarrow 0) \rightarrow ((1 \rightarrow 0) \rightarrow 1) \).

3.2 NOTATION. Greek letters will denote arbitrary types. Parentheses will be omitted in such a way that, for example,

\[ \alpha \rightarrow \beta \rightarrow \gamma \rightarrow \delta = (\alpha \rightarrow (\beta \rightarrow (\gamma \rightarrow \delta))) \]

3.3 DEFINITION (HS 13.3): *typed \( \lambda \)-terms*. For each type \( \alpha \), we assume an infinite set of *typed variables* \( x^{\alpha}, y^{\alpha}, \ldots \)

(a) Each typed variable is a typed term;

(b) from \( x^{\alpha \rightarrow \beta}, y^{\beta} \), construct a new typed term \( (x^{\alpha \rightarrow \beta} y^{\beta})^{\beta} \);

(c) from \( x^{\alpha}, y^{\beta} \), construct a new typed term \( (\lambda x^{\alpha} y^{\beta})^{\alpha \rightarrow \beta} \).

The type-superscript on the right of the term is called the term’s type.

3.4 EXAMPLES. The following typed terms have special names.

\( I_{\alpha} = (\lambda x^{\alpha}. x^{\alpha})^{\alpha \rightarrow \alpha} \) (there is one \( I_{\alpha} \) for each \( \alpha \)).

\[ K_{\alpha, \beta} = (\lambda x^{\alpha}. y^{\alpha} x^{\alpha})^{\alpha \rightarrow \beta \rightarrow \alpha} \] (one \( K_{\alpha, \beta} \) for each \( \alpha, \beta \)).

The construction of \( K_{\alpha, \beta} \) is shown by the following tree:

\[
\begin{array}{c}
\lambda x^{\alpha} \\
\text{(there is one \( I_{\alpha} \) for each \( \alpha \)).}
\end{array}
\]

\[
\begin{array}{c}
\lambda y^{\beta}. x^{\alpha} \\
\text{(one \( K_{\alpha, \beta} \) for each \( \alpha, \beta \)).}
\end{array}
\]

\[
\begin{array}{c}
\lambda x^{\alpha}. y^{\alpha} x^{\alpha} \\
\text{\( \rightarrow \beta \rightarrow \alpha \).}
\end{array}
\]

\[
\begin{array}{c}
\lambda x^{\alpha}. y^{\beta}. z^{\alpha}. x z(y z) \text{\( \rightarrow \gamma \rightarrow \beta \rightarrow \alpha \rightarrow \gamma \).}
\end{array}
\]
The construction-tree of $S_{\alpha, \beta, \gamma}$ is (with some type-superscripts omitted):

$$
\begin{align*}
\frac{x^{\alpha} \rightarrow \beta \rightarrow \gamma \quad z^{\alpha}}{(xz)^{\beta}} & \quad \frac{y^{\alpha} \rightarrow \beta \rightarrow \gamma}{(yz)^{\beta}} \\
\frac{((xz)(yz))^{\gamma}}{(\lambda z.xz(yz))^{\alpha \rightarrow \gamma}} & \quad \frac{((\lambda y.zx(xz(yz))^{\alpha \rightarrow \beta}) \rightarrow \alpha \rightarrow \gamma}{(\lambda y.zx(xz(yz))^{\alpha \rightarrow \beta} \rightarrow \alpha \rightarrow \gamma}
\end{align*}
$$

3.5 NOTE. Substitution, $[N/x]$, is defined for typed terms exactly as for untyped terms, except that now $N$ and $x$ must have the same type.

Reduction and equality are defined exactly as for untyped terms (HS 13.9 - 10). The Church-Rosser theorems and other principal results remain true (HS 13.11), plus the following.

3.6 STRONG NORMALIZATION THEOREM (HS 13.12). Every $\beta$-reduction starting with a typed term is finite.

3.6.1 COROLLARY (NORMAL-FORM THEOREM). Every typed term has a $\beta$-normal form.

3.6.2 COROLLARY. The relations $*_{\beta}$ and $\triangleright_{\beta}$, restricted to typed terms, are recursively decidable.

4. TYPED COMBINATORY LOGIC

4.1 DEFINITION (HS 13.16): typed CL-terms. For each type $\alpha$, assume an infinite sequence of typed variables $x^{\alpha}, y^{\alpha}, \ldots$. For each $\alpha, \beta$, assume distinct typed basic combinators $K^{\alpha \rightarrow \beta \rightarrow \alpha}, S^{\alpha \rightarrow \beta \rightarrow \gamma}(\alpha \rightarrow \beta \rightarrow \alpha \rightarrow \gamma)$.

(a) These variables and basic combinators are typed CL-terms;

(b) from $x^{\alpha \rightarrow \beta}, y^{\alpha}$, construct a new typed CL-term $(x^{\alpha \rightarrow \beta}y^{\alpha})^{\beta}$.

The type of a typed term is the type-superscript on the right.
4.2 **DEFINITION (HS 13.18):** **Typed weak reduction.** This is defined exactly as in 2.4, by contractions of form

\[(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \gamma \rightarrow \alpha \rightarrow \beta, \text{ } p_w \rightarrow \alpha \rightarrow \beta, \text{ } p_w \rightarrow (\gamma \rightarrow \alpha \rightarrow \beta, \text{ } p_w \rightarrow (\gamma \rightarrow \alpha \rightarrow \beta, \text{ } p_w \rightarrow (\gamma \rightarrow \alpha \rightarrow \beta).

4.3 **NOTE.** Abstraction, \(\lambda^x.M\), is defined exactly as in 2.8. The Church-Rosser theorems and other main properties of weak reduction and abstraction (HS 2.7 - 24) remain true. We also have a strong normalization theorem and a normal-form theorem for typed weak reduction, just like Theorem 3.6 and its corollaries. (HS 13.22.)

5. **TYPE-ASSIGNMENT IN COMBINATORY LOGIC**

Typed terms correspond to the concept of function that is standard in mathematics, but not in computation theory. In the latter, a function may be defined without first defining its domain. (See HS 3.24.) Such functions are best represented by untyped terms; but we can assign to each term a set of expressions called **type-schemes**, which show the relation of range to domain.

5.1 **DEFINITION (HS14.1):** **type-schemes.** Choose some atomic **type-constants** (e.g. 0, 1) to denote particular chosen sets, plus an infinite number of **type-variables** \(a, b, c, \ldots\), to denote arbitrary sets. Then

(a) each type-constant or type-variable is a type-scheme;
(b) from any type-schemes \(\alpha, \beta\), build a new type-scheme \((\alpha \rightarrow \beta)\).

**Examples:** \(0 \rightarrow a \rightarrow 0, \text{ } a \rightarrow b \rightarrow a.

**NOTATION.** The same notation will be used for type-schemes as for types. (Every type is in fact a type-scheme.)

**Warning:** the Greek letters '\(\alpha\)', '\(\beta\)' are not type-variables, and the expression '\((\alpha \rightarrow \beta)\)' is not a type-scheme. It is merely an expression in the meta-language, and comes to denote a type-scheme when its Greek letters are interpreted as particular type-schemes.

A **combinator** is any term whose only atoms are \(K\) and \(S\).
5.2 DEFINITION (HS 14.4). A type-assignment formula is any expression of form \( X \in \alpha \), where \( X \) is an untyped term (called the subject), and \( \alpha \) is a type-scheme (called the predicate).

**Interpretation:** If \( \alpha \) and \( \beta \) denote sets, then a formula \( X \in \alpha \to \beta \) means that \( X \) is a function which assigns to each member of \( \alpha \), at most one member of \( \beta \). Note that \( XY \) may be undefined for some \( Y \) in \( \alpha \), and defined for some \( Y \) not in \( \alpha \).

5.3 DEFINITION (HS 14.5). The type-assignment system \( TA_\alpha \) is a formal system with 2 axiom-schemes and one rule of inference:

Axiom-schemes:
- \((\to K)\): \( K \in \alpha \to \beta \to \alpha \),
- \((\to S)\): \( S \in (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma \).

Rule \((\to e)\), or \(-\) elimination: \( X \in (\alpha \to \beta), Y \in \alpha \to (XY) \in \beta \).

5.4 NOTATION. An axiom is a special case of an axiom-scheme, for example \( K \in 0 \to (1 \to a) \to 0 \). Rule \((\to e)\) says that we can deduce \( XY \in \beta \) from formulas \( X \in (\alpha \to \beta) \) and \( Y \in \alpha \). Iff a formula \( X \in \alpha \) is deducible from the axioms by the rule, we say \( \vdash X \in \alpha \).

Iff \( X \in \alpha \) is deducible from axioms and extra assumptions \( x_1 \in \delta_1, \ldots, x_n \in \delta_n \), where \( x_1, \ldots, x_n \) are the distinct variables in \( X \), we say \( x_1 \in \delta_1, \ldots, x_n \in \delta_n \vdash X \in \alpha \).

A variables-subjects basis is any set of formulas of form \( x_1 \in \delta_1, \ldots, x_n \in \delta_n \).

5.5 EXAMPLE. For all \( \alpha \), we have \( \vdash SKK \in \alpha + \alpha \).

**Proof:** let \( \beta = \alpha + \alpha \), \( \gamma = \alpha \);

\[
\begin{align*}
S & \in (\alpha + \beta + \alpha) + (\alpha + \beta) + \alpha + \alpha & K & \in \beta + \alpha + \alpha \\
(\to e) & \quad (SK) \in (\alpha + \beta) + \alpha + \alpha & K & \in \alpha + (\alpha + \alpha) \\
& \quad SKK \in \alpha + \alpha .
\end{align*}
\]

5.6 NOTE. The form of this deduction is dictated by the structure of the term \( SKK \), and the whole deduction corresponds
exactly to the typed term
\[
((\beta \cdot \alpha) \cdot (\beta \cdot \alpha) \cdot \alpha) \cdot \alpha \cdot \alpha \cdot \alpha \cdot \alpha \cdot \alpha \cdot \alpha.
\]

5.7 Definition (HS 14.28). A combinator \( X \) is stratified if there exists \( \alpha \) such that \( \vdash X \in \alpha \). A term \( X \), whose only atoms are \( S, K \) and variables, is stratified if there exist \( \alpha \) and a variables-subjects basis \( B \) such that
\[
B \vdash X \in \alpha.
\]

5.8 Exercises. Show that, if \( B = S(KS)K \) and \( C = S(BBS)(KK) \),
(a) \( \vdash B \in (\beta \cdot \gamma) \cdot (\alpha \cdot \beta) \cdot \alpha \cdot \gamma \),
(b) \( \vdash C \in (\alpha \cdot \beta \cdot \gamma) \cdot \beta \cdot \alpha \cdot \gamma \),
(c) \( \lambda \cdot x . x x \) is stratified,
(d) \( \lambda \cdot x . x x \) is not stratified.

5.9 Abstraction-and-Types Theorem (HS 14.19). If \( x_1, \ldots, x_n, x \) are the variables in \( X \), and
\[
x_1 \in \delta_1, \ldots, x_n \in \delta_n, x \in \alpha \vdash X \in \beta,
\]
then
\[
x_1 \in \delta_1, \ldots, x_n \in \delta_n \vdash (\lambda \cdot x . x) \in (\alpha \rightarrow \beta).
\]

5.10 Subject-Reduction Theorem (HS 14.24). If \( B \) is a variables-subjects basis and \( B \vdash X \in \alpha \) and \( X \equiv x' \), then
\[
B \vdash x' \in \alpha.
\]

5.11 Strong Normalization Theorem (HS 14.42). If \( X \) is stratified, then all weak reductions starting at \( X \) are finite.

5.11.1 Corollary. Every stratified term has a weak normal form.

5.12 Theorem (HS 14.41). The set of all stratified terms is recursively decidable.

5.13 Definition (HS 14.31). A principal type-scheme (pts) of a stratified combinator \( X \) is a type-scheme \( \alpha \) such that
(a) \( \vdash X \in \alpha \),
(b) if \( \vdash X \in \beta \), then \( \beta \) is a substitution-instance of \( \alpha \).

Example. A pts of \( SKK \) is \( a \rightarrow a \). (Another is \( b \rightarrow b \).)
5.14 THE PTS THEOREM (HS 14.40). Every stratified combinator has a pta.


5.15 REMARK (HS 14.46). If we take a $Th_C$-deduction, for example the one for SKK in Example 5.5, and delete all the type-schemes, then we get a tree

\[
SKK
\]

which is the construction-tree for SKK. In contrast, if we delete all the terms instead, we get a tree

\[
\frac{(\alpha+\beta+\alpha)+\alpha+\alpha+\alpha+\alpha}{(\alpha+\beta)+\alpha+\alpha} \quad (\beta \equiv \alpha+\alpha)
\]

This tree is a deduction in intuitionist implicational logic. In fact, the type-schemes of the stratified combinators correspond exactly to the provable formulas of intuitionist implicational logic.

6. TYPE-ASSIGNMENT IN LAMBDA-CALCULUS

6.1 DEFINITION (HS 15.1-6). The type-assignment system $TA_{\Lambda}$ is like Gerhard Gentzen's 'Natural Deduction' systems in logic (see Prawitz 1965). It has no axioms, and has the following 3 rules, called ($\rightarrow e$) or $\rightarrow -$elimination, ($\rightarrow i$) or $\rightarrow -$introduction, and ($\alpha$):

\[
\frac{M \in (\alpha+\beta) \quad N \in \alpha}{(MW) \in \beta} \quad [x \in \alpha] \quad (\rightarrow e)
\]

\[
\frac{M \in (\alpha+\beta) \quad N \in \beta}{M = N \in \alpha} \quad (\rightarrow i)
\]

\[
\frac{M \in \beta \quad \alpha \vdash N}{(\alpha x.M) \in (\alpha+\beta)}.\]

Explanation. Rule ($\rightarrow i$) means that if we already have a deduction of a formula $M \in \beta$ from $x \in \alpha$ and perhaps a set $\beta$ of other assumptions, then we can deduce, from $\beta$ alone, the formula

\[
(\lambda x.M) \in (\alpha+\beta).
\]

Rule ($\rightarrow i$) is not permitted to be used if $\beta$ contains $x$.

Also, after ($\rightarrow i$) has been used, we enclose the formula $x \in \alpha$
in brackets wherever it occurs, to show that it is now no longer regarded as an assumption. It is now called a cancelled, or discharged, assumption. (Cf. HS 15.1.) So a deduction grows in two ways, by adding new conclusions at the bottom, and by adding brackets to assumptions.

6.2 DEFINITION: deducibility. Iff there exists a deduction of $X \in \alpha$ with all assumptions cancelled, we say

$$\vdash X \in \alpha.$$  

Iff there is a deduction with all assumptions cancelled except those in $B$, we say

$$B \vdash X \in \alpha.$$  

6.3 EXAMPLE. $\vdash (\lambda xyz.xz(yz)) \in (\alpha+\beta+\gamma)+(\alpha+\beta)+\alpha+\gamma.$

Proof. See below. Each of the assumptions is uncancelled at the start of the proof, and then becomes cancelled later. Compare the binding of a variable by a $\lambda$. In fact, the proof below is exactly like the construction-tree for $\lambda x, y, z$ in Example 3.4.

6.4 EXAMPLE. $\vdash (\lambda xy.x) \in \alpha+\beta+\alpha$.

Proof. In this proof, an assumption $y \in \beta$ is cancelled, even though it has not actually been used. This is permitted.

6.5 EXAMPLE. The following is not a deduction, because when rule $\rightarrow i$ is first applied, cancelling $x \in \alpha$, a statement containing $x$ also appears in the uncancelled set.
The theorems stated for TAλ, 5.10 - 15, hold true also for TAₜ.
For more information on TAₜ, see HS Chapter 15.

BIBLIOGRAPHY


Church, A. [1941] The Calculi of Lambda Conversion, Princeton Univ. Press; reprinted 1963 by University Microfilms Inc., Ann Arbor, Michigan, U.S.A.


