A mild variant of Combinatory Logic.

The following is written in reaction to the description of Combinatory Logic as given in [1] by D.A.Turner. With plus standing for the prefix adding operator, Turner defines the successor function suc by

$$\underline{\text{def}}\ \text{suc}\ x = \underline{\text{plus}}\ 1\ x \quad .$$

To the right of the equality sign, juxtaposition is assumed to be associative to the left; more fully bracketed we would have had

$$\underline{\text{def}}\ \text{suc}\ x = (\underline{\text{plus}}\ 1)\ x \quad .$$

Because at both sides of the equality sign, the variable x occurs only as the very last atom, we can abstract x from both sides by omission, yielding

$$\underline{\text{def}}\ \text{suc} = \underline{\text{plus}}\ 1 \quad .$$

This wouldn't have been possible if suc had been defined by

$$\underline{\text{def}}\ \text{suc}\ x = (\underline{\text{plus}}\ x)\ 1$$

because here the right-hand side is not an expression in which x --i.e. the variable to be abstracted from-- occurs only as the very last atom.

Combinatory logic is a discipline for rewriting, if necessary, such right-hand sides in such a way that x --or, in general: the variable to be abstracted from-- only occurs as the last atom of the expression.
Given

$$\underline{\text{def}}\ f\ x = E$$

we derive a G , such that

$$G\ x = E \quad \text{(Law of Abstraction)}$$

from which "by omission"

$$\underline{\text{def}}\ f = G$$

then follows.

Turner denotes G by $[x]E$ or by $([x]E)$ --to be pronounced as "abstract x from E"-- and describes (following Curry) abstraction as a textual operation. I almost quote from [1] --remember that juxtaposition associates to the left, at least at both sides of the equality sign--

"We introduce three combinators S , K, and I [It is customary to print these constants in bold face, EWD] , defined by the equations

$$S\ f\ g\ x = f\ x\ (g\ x) \qquad (S)$$
$$K\ y\ x = y \qquad (K)$$
$$I\ x = x \qquad (I)$$

and define abstraction as follows

$$[x](E1\ E2) = S\ ([x]E1)\ ([x]E2)$$
$$[x]x = I$$
$$[x]y = K\ y$$

where y is a constant or a variable other than x . That the abstraction thus defined obeys the Law of Abstraction given above can be proved by an induction on the size of the combination E -- we leave this as an exercise for the interested reader."

After having pointed out that this abstraction process leads to very long-winded formulae, Turner introduces for brevity's sake two extra combinators B and C , defined by

$$B\ f\ g\ x = f\ (g\ x) \qquad (B)$$
$$C\ f\ g\ x = f\ x\ g \quad ; \qquad (C)$$

please remember the left-associative juxtaposition: we could have written $((C\ f)\ g)\ x = (f\ x)\ g$.

*   *   *

So much for Turner's description of combinatory logic. His notational conventions have the (potential) technical advantage that each formula can be represented by a binary tree with labels --combinators or identifiers-- in its leaves.

I found the fact that during the abstraction process combinators and identifiers (for variables) are treated on the same footing more confusing than illuminating. As it stands, the abstraction process also defines how to abstract a combinator from an expression; perhaps somewhat rashly I regard this, however, as a rather meaningless facility.

Note. Not only some abstractions, but also some applications should be regarded as meaningless. I owe to C.S.Scholten the following very nasty example. From

    def funny x = x x

we derive according to the rules

    x x = S I I x

hence

    def funny = S I I .

How well-deserved the name "funny" is is demonstrated by applying it to itself and reducing:

    funny funny = S I I (S I I) =
    I (S I I) (I (S I I)) =
    S I I (S I I) = funny funny .

The expression (funny funny) is endlessly reducible! (End of Note.)

I was also disturbed by the extreme asymmetry introduced by the combination of a) the placing of the combinators and b) the association to the left. This is particularly noticeable in the fully bracketed form of rule (S):

    ((S f) g) x = (f x) (g x) .

To the right we have two expressions, both containing the variable x . The combinator S , though pertaining in a way to both, is placed as a prefix to the left-hand one. My gut feeling --which of course may err-- tells me that this is ugly.

Note. It could be argued that application is not only asymmetric --in general "a b $\neq$ b a"-- but that the two components are very different, being "function" and "argument" respectively. But two remarks are in order.

Firstly, the distinction between "function" and "argument" is not as clear-cut as is often thought. With z standing for a complex number and re(z) and im(z) standing for its real and imaginary parts respectively, we could regard re and im as real functions defined on a complex domain. With z as a record with two fields we would write z.re and z.im respectively, i.e. a complex value can be regarded as a real function on the domain {re, im}.

Secondly, it is exactly combinatory logic that allows expressions for a and b , such that "a b" and "b a" , though different, are extremely meaningful. (See below.) (End of Note.)

A third complaint only emerged when exercising it. While abstracting and applying I found myself all the time adding and removing bracket pairs. This is not to be wondered at: when we look at the S-rule in its fully bracketed form, it is not clear at all, how the bracket pairs to the right and to the left correspond with each other.

So I set myself the goal of finding an equivalent notation to which the above objections would not apply. Most of the exploratory work was done together with C.S.Scholten; the synthesis to be described below emerged on 8 April 1980 in a session of the Tuesday Afternoon Club, which screened the manuscript for what follows a week later.

* * *

In the following there is no such thing as association from the left, and I shall introduce no redundant or optional bracket pairs. My starting point is the set of fully bracketed expressions, satisfying the syntax

$$< \text{exp} > ::= < \text{identifier} > \mid (< \text{exp} >< \text{exp} >) \quad .$$

I shall not introduce the need to abstract a variable from an expression in which that variable does not occur. The formal definition for occurrence is the following obvious one: with x and y standing for identifiers and E, E1, and E2 standing for expression, we define

```
x in E = if E = y → x = y
            ▯ E = (E1 E2) → (x in E1) or (x in E2)
          fi   .
```

We shall now define how to abstract an identifier x from an expression E , provided x in E . We shall do so by defining abstraction from a more general class than just expressions, viz. "terms". While expressions are built from parentheses and identifiers only, terms may also contain so-called combinators; a term may also be empty. In the following coms stands for a possibly empty sequence of combinators, and coms+ for a non-empty sequence of combinators. The syntax for terms is (see, however, correction on next page)

$$< \text{term} > ::= < \text{empty} > \mid < \text{exp} > \mid (< \text{term} > \text{coms+} < \text{term} >) \quad .$$

Note that each bracket pair always surrounds two terms, separated by a possibly empty sequence of combinators, such that neither of the terms is empty or the combinator sequence is non-empty. I shall not introduce the need to abstract a variable from a term in which that variable does not occur; with furthermore T , T1 , and T2 standing for terms, we define

```
x in T = if T = empty → false
         ▯ T = E → x in E
         ▯ T = (T1 coms+ T2) → (x in T1) or (x in T2)
         fi        .
```

Provided x in T , "abstract x from T " --as usual denoted by "[x]T"-- is defined by

```
[x]T = if x in T →                                        0.
          if T = x → empty                                1.
           ▯ T = (T1 coms T2) →                           2.
             if x in T1 →                                 3.
               if x in T2 → ([x]T1 S coms [x]T2)          4.
                ▯ non x in T2 → ([x]T1 C coms T2)         5.
               fi                                         6.
             ▯ non x in T1 → {x in T2}                    7.
               if T = (T1 x) → T1                         8.
                ▯ T ≠ (T1 x) → (T1 B coms [x]T2)          9.
               fi
             fi
          fi
       fi      .
```

Notes. The guard in line 8 is only true if coms is the empty sequence of combinators and T2 = x . This line represents the only case in which an original bracket pair disappears. No new bracket pairs are introduced; hence, the resulting bracket pairs are a subset of the original ones. Line 8 describes "abstraction by omission".

Because in line 2 coms = < empty > implies that neither T1 nor T2 is empty, line 4 represents the only possibility of generating a bracket pair containing a single atom: (S) . Therefore, in line 8 T1 is not empty. (End of Notes.)

A reduction rule is applicable whenever we encounter a term of the form

((T1 coms+ T2) h) ;

in view of the rule of the absence of redundant parentheses this implies that **h** is not the empty term. The nature of the reduction is determined by the leading combinator of coms+ :

((T1 S coms T2) h) = ([T1 h] coms [T2 h])
((T1 C coms T2) h) = ([T1 h] coms T2)
((T1 B coms T2) h) = (T1 coms [T2 h])

where [T h] is short for

$[T\ h] = \underline{\text{if}}\ T = <\text{empty}> \rightarrow h\ [\!]\ T \neq <\text{empty}> \rightarrow (T\ h)\ \underline{\text{fi}}$.

Like Turner I leave the verification that

[[x]T x] = T (Law of Abstraction)

as an exercise for the interested reader.

Note. We have already observed that in a term derived by abstraction from an expression and of the form (T1 S T2) both T1 and T2 may be empty. If it is of the form (T1 B T2), T1 will not be empty; if it is of the form (T1 C T2), T2 will not be empty. As a result, reduction will never lead to a redundant bracket pair. (End of Note.)

* * *

Correction. The syntax for < term > as given previously is too restrictive: according to it the above ((T1 coms+ T2) h) is not a term! The syntax

< term > ::= < empty > | < exp > | (< term > coms < term >)

is too broad; it is acceptable provided we rule out from the last alternative the forms (), (< term > C), and (B < term >) , or state the non-occurrence of such terms as a theorem. See previous note. (End of Correction.)
NB. See Appendix on page EWD735 - 8, etc.!!!!!

Of course the new notation doesn't rule out anomalies like funny. We would have

$$\underline{\text{def}}\ \text{funny} = (S) \quad .$$

According to the new rules we find in a single reduction step --the intermediate stage with the square brackets has only been given for the sake of completeness--

$$((S)(S)) = ([(S)][(S)]) = ((S)(S)) \quad .$$

As another example we consider the function twice , given by

$$\begin{aligned} ((\text{twice } f)\ x) &= (f\ (f\ x)) &&, \text{hence} \\ (\text{twice } f) &= (f\ B\ f) &&, \text{hence} \\ \text{twice} &= (SB) \end{aligned}$$

and similarly the function thrice , given by

$$\begin{aligned} ((\text{thrice } f)\ x) &= (f\ (f\ (f\ x))) &&, \text{hence} \\ (\text{thrice } f) &= (f\ B\ (f\ B\ f)) &&, \text{hence} \\ \text{thrice} &= (SB\ (SB)) &&. \end{aligned}$$

Now we consider and reduce as far as possible

$$\begin{aligned} (\text{thrice twice}) &= ((SB\ (SB))\ \text{twice}) \\ &= (\text{twice } B\ (\text{twice } B \text{ twice})) \\ &= ((SB)\ B\ ((SB)\ B\ (SB))) \quad . \end{aligned}$$

Hence

$$\begin{aligned} ((\text{thrice twice})\ f) &= (((SB)\ B\ ((SB)\ B\ (SB)))\ f) \\ &= ((SB)\ ((SB)\ (f\ B\ f))) \\ &= ((SB)\ ((f\ B\ f)\ B\ (f\ B\ f))) \\ &= (((f\ B\ f)\ B\ (\ f\ B\ f))\ B\ ((f\ B\ f)\ B\ (f\ B\ f))) \quad . \end{aligned}$$

Hence $(((\text{thrice twice})\ f)\ x) = (f\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ x))))))))$ i.e. $f^8(x)$ --where I placed the brackets in the traditional fashion!--

As a final example, consider $E = ((*\ ((+\ x)\ y))\ y)$
I then derive without further intermediate results

$$\begin{aligned} [x]E &= ((*\ B\ (+\ C\ y))\ C\ y) \\ [y][x]E &= ((*\ BB\ (+\ BC))\ SC) \end{aligned}$$

In Turner's style we would have had $E = *\ (+\ x\ y)\ y$.

In that case I had to derive

$$\begin{aligned}[x]E &= C\ [x](*\ (+\ x\ y))\ y \\ &= C\ (B\ *\ (C\ [x](+\ x)\ y))\ y \\ &= C\ (B\ *\ (C\ +\ y))\ y \qquad .\end{aligned}$$

(Actually I had one intermediate step more.) Skipping the intermediate steps I found

$[y][x]E = S\ (B\ C\ B\ (B\ *)\ (C\ +))\ I$, or fully bracketed --almost fully, that is--
$= (S\ ((((B\ C)\ B)\ (B\ *))\ (C\ +)))\ I$

*   *   *

The difference between the two techniques is clearly that I propose to attach strings of combinators to internal nodes of the parse tree; the other convention expands the binary tree and stores the information of the strings of combinators in the leaves thus created. I haven't learned yet, how to translate trees of the one type into trees of the other type. I cannot escape the impression that Curry's conventions for "tree expansion" are in some sense more arbitrary than the conventions I have explored here.

Another moral of the story is that "Currying" as introduced by Turner is from a logical point of view no more than another red herring. Why replace "x + y" first by "<u>plus</u> x y", when we can parse these formulae as "(x +) y" and "(<u>plus</u> x) y" ? The <u>only</u> justification is that [x](<u>plus</u> x) = <u>plus</u> is shorter than [x](x +) = (C I) + --according to Curry/Turner-- or (C +) --according to the conventions explored here-- : in short, no more than a minor optimization if we assume that x is more frequently abstracted from (x +) than + .

I have the feeling that the above explorations should be continued.

NB. Appendix on the next pages!!!! (End of NB.)

Plataanstraat 5	23 April 1980
5671 AL NUENEN	prof.dr.Edsger W.Dijkstra
The Netherlands	Burroughs Research Fellow

Appendix. (Written after the above had been read together with C.S.Scholten.)

A first remark is that we can simplify the abstraction process as described on page EWD735 - 4 by replacing the alternative construct on lines 8, 9, and 10 simply by (T1 B coms [x]T2) . Cleaning things up we get

```
[x]T = if x in T →                                                   .0
          if T = x → empty                                           .1
           ▯ T = (T1 coms T2) →
             if x in T1  and  x in T2 → ([x]T1 S coms [x]T2)
              ▯ x in T1  and  non x in T2 → ([x]T1 C coms T2)
              ▯ non x in T1  and  x in T2 → (T1 B coms [x]T2)
             fi
          fi
       fi    .
```

From a theoretical point of view, this has two advantages. Firstly, the bracket pairs in the resulting term are now the same bracket pairs that occurred in the original term. Secondly, "abstraction by omission" has now been confined to line 1 , and in (T1 coms T2) the constituent terms T1 and T2 are treated on equal footing. Line 8 in the original version is no more than saving symbols at the price of symmetry, and that is at this level of discussion a doubtful procedure.

A more important remark is that my correction on page EWD735 - 5 was a short-sighted patch. We should have defined the syntax for terms independently of that for expression, viz.

< term > ::= < empty > | < identifier > | (< term > coms < term >) .

This syntax does not rule out redundant bracket pairs. When we now define "T is healthy" to mean:

T is empty or

T is an identifier or

T is of the form (T1 coms T2) such that

(T1 $\neq$ empty or coms contains an S or a C) and

(T2 $\neq$ empty or coms contains an S or a B) and

T1 and T2 are both healthy

we can observe to start with that each expression is a healthy term. Furthermore no healthy term contains a redundant bracket pair --either of the form "()" or of the form "((...))"-- .

Furthermore --both in the original form of abstraction and in the cleaned up one-- a healthy T implies (when defined) a healthy [x]T . Furthermore , for healthy T and healthy, but non-empty h , [T h] as defined on page EWD735 - 5 is healthy. In short, neither abstraction nor reduction ever introduces an unhealthy term!

Salvo errore et omissione, the above does the job. By the time that we have discovered the significance --if any-- of our exercise, this text has to be rewritten anyhow; hence I think that this rough "working document" has reached the stage to be concluded. Besides that, I am tired.

* * *

At this stage it is very hard to formulate my acknowledgments. I am indebted to D.A.Turner for his article, to Hamilton Richards for having drawn my attention to it; I am greatly indebted to C.S.Scholten for joining me in my effort to understand combinatory logic in my way: his assistance in all stages of the development has been invaluable; I am indebted to the members of the Tuesday Afternoon Club for their help and criticism. The initiative and the errors are mine.

(End of Appendix.)

[1] Turner, D.A., "A New Implementation Technique for Applicative Languages." Software--Practice and Experience, Vol.9, 31-49 (1979).

Plataanstraat 5
5671 AL NUENEN
The Netherlands

6 May 1980
prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow