Highlights of the History of the Lambda-Calculus

J. BARKLEY ROSSER

This paper gives an account of both the lambda-calculus and its close relative, the combinatory calculus, and explains why they are of such importance for computer software. The account includes the shortest and simplest proof of the Church-Rosser theorem, which appeared in a limited printing in August 1982. It includes a model of the combinatory calculus, also available in 1982 in a limited printing. In the last half-dozen years, some revolutionary new ideas for programming have appeared, involving the very fundamentals of the lambda-calculus and the combinatory calculus. A short introduction is given for a couple of these new ideas.

*Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—computability theory; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—lambda-calculus and related systems; K.2 [**History of Computing**]—people, software*

General Terms: Theory

Additional Key Words and Phrases: combinatory calculus, Turing machines, foundations of programming, Church's thesis

"Kleene-ness is next to Gödel-ness"

1. Early Beginnings

The lambda-calculus originated in order to study functions more carefully. In 1893 Frege observed (see van Heijenoort 1967, p. 355) that it suffices to restrict attention to functions of a single argument. Suppose we wish a function to apply to A and B to produce their sum, $A + B$. Let $\oplus$ be a function, of a single argument, that when applied to A alone produces a new function, again of a single argument, whose value is $A + B$ when applied to B alone. Note that $\oplus$ is not applied simultaneously to A and B, but successively to A and then B; application to A alone produces an intermediary function $\oplus(A)$, which gives $A + B$ when later applied to B alone. That is, $A + B = (\oplus(A))(B)$.

This method of reducing the use of a function "+" of two arguments to proper use of a related function "$\oplus$" of one argument only is often referred to as "currying" because it was brought into prominence by the writings of Haskell B. Curry. Obviously, the method can be extended to reduce the use of a function of still more arguments to proper use of a related function of one argument only.

This is the way computers function. A program in a computer is a function of a single argument. People who have not considered the matter carefully may think, when they write a subroutine to add two numbers, that they have produced a program that is a function of two arguments. But what happens when the program begins to run, to produce the sum $A + B$? First A is brought from memory. Suppose that at that instant the computer is completely halted.

Adapted with permission from a paper presented at the 1982 ACM Symposium on LISP and Functional Programming. © 1982, Association for Computing Machinery, Inc. Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

Author's Address: Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, WI 53705.

© 1984 AFIPS 0164-1239/84/040337-349$01.00/00

What remains in the computer is a program, to be applied to any B that might be forthcoming, to produce the sum of the given A and the forthcoming B. It is a function of one argument, depending on the given A, to be applied to any B, to produce the sum $A + B$. It is Frege's intermediary function $\oplus(A)$.

Apparently Frege did not pursue the idea further. It was rediscovered independently (see Schönfinkel 1924), together with the astonishing conclusion that all functions having to do with the structure of functions can be built up out of only two basic functions, K and S. Let us adopt the notation that has been in vogue since then. Instead of writing the value that we get by applying the function F to A as $F(A)$, we write (FA). Omission of the outside parentheses will be usual. When more than two terms occur, association is to the left; thus MNP denotes $((MN)P)$, but $M(NP)$ denotes $(M(NP))$. Then the sum of A and B would be written $\oplus AB$.

The functions K and S are such that

$$KAB = A \tag{1.1}$$

$$SABC = AC(BC) \tag{1.2}$$

For a proof that all functions can be built up of K and S, we can consult the original Schönfinkel paper, two early papers by Curry (1929 or 1930), or Curry and Feys (1958, pp. 186–189).

Expressions built up out of K and S by application (that is, enclosing pairs in parentheses) are called "combinators." Use of them, and study of their properties, is called "combinatory logic." Sometimes these labels are extended to apply when the expressions are allowed to contain variables, or indeterminates, as well as K and S.

Suppose we have two functions F and G (built up out of K and S, of course) such that by means of (1.1) and (1.2) we can show that

$$FX = GX \tag{1.3}$$

J. Barkley Rosser was born in Jacksonville, Florida, in 1907. He received a Ph.D. from Princeton University in 1934, and was a fellow graduate student with Stephen C. Kleene under Alonzo Church when the latter was first presenting the lambda-calculus. Rosser was president of the Association for Symbolic Logic in 1950–1953.

for each X, or for an indeterminate X. Accordingly, F and G take the same value whenever they are applied to each X whatever, and so they ought to be the same function. That is, we should have

$$F = G \tag{1.4}$$

In general, we cannot prove (1.4) by means of (1.1) and (1.2). Curry (1930) contrived additional axioms such that we can prove (1.4) whenever (1.3) holds for each X. A system like this is said to have the extensional property.

Curry added axioms to enable him to prove additional equalities. Did he go too far, so that now any two functions can be proved equal to each other? He did not. Indeed, he was careful to prove a weak form of consistency, in that many pairs of functions cannot be proved equal to each other; especially, $K = S$ cannot be proved.

The system was workable, and illuminated many properties of functions. For instance, let M be built up from K, S, and the variable x. One can, using only K and S, build up a function F such that one can prove that

$$Fx = M$$

by means of (1.1) and (1.2). F turns out to be a mixed-up combination of K's and S's. Just from looking at F, one would not have the least clue that $Fx = M$ should hold.

Because F is constructed in order to give the result $Fx = M$, it follows that

$$FN = M[x{:=}N] \tag{1.5}$$

in which $M[x{:=}N]$ means the result of replacing each occurrence of x in M by N.

Alonzo Church (1932) proposed that the F in question be called $(\lambda x(M))$, commonly abbreviated to λxM. Here, M is intentionally part of the name of the function, so that by inspection we can see what we would get if we apply the function to x. For his construct, Church decreed that

$$(\lambda xM)N = M[x{:=}N] \tag{1.6}$$

which accords exactly with (1.5).

Church was struck with certain similarities between his new concept and that used in Whitehead and Russell (1925) for the class of all x's such that $f(x)$; to wit, $\hat{x}f(x)$. Because the new concept differed quite appreciably from class membership, Church moved the caret from over the x down to the line just to the left of the x; specifically, $\wedge xf(x)$. Later, for reasons of typography, an appendage was added to the caret to produce a lambda; the result was $\lambda xf(x)$.

Starting with the left side of (1.6) and replacing it by the right side is called a β-reduction. We are equally entitled to start with the right side of (1.6) and replace it by the left side of (1.6)—a β-expansion.

Thus, to produce the $\oplus$ that we had earlier, Church would use $\lambda x(\lambda y(x + y))$. By (1.6), we have

$$(\lambda x(\lambda y(x + y)))A = \lambda y(A + y) \tag{1.7}$$

By (1.6) again, we have

$$(\lambda y(A + y))B = A + B \tag{1.8}$$

So, taking $\lambda x(\lambda y(x + y))$ to be $\oplus$, we have

$$\oplus AB = (\lambda x(\lambda y(x + y)))AB = A + B \tag{1.9}$$

by two β-reductions.

The beauty of these manipulations is that at all stages of the process, we can tell by a simple inspection what the reduced form of $\oplus AB$ is going to be. What we have been describing is the famous lambda-calculus of Church. (Henceforth, we will write LC for "lambda-calculus.") We do have to be careful about free and bound occurrences of variables. In $\lambda y(x + y)$, the occurrence of x is free, and both occurrences of y are bound.

We have to be careful not to make manipulations that change free occurrences of a variable into bound ones. Thus, suppose we write $\oplus y$. In this configuration, the observed occurrence of y is free. Blind adherence to (1.6) would give

$$\oplus y = \lambda y(y + y)$$

so that

$$\oplus yz = z + z$$

This equation is certainly not what is intended for $\oplus$. The trouble is that the y, which originally existed as a free occurrence of a variable in $\oplus y$, has been put into $\lambda y(y + y)$ where its occurrence is now bound. Actually, when Church enunciated the rule (1.6), he was careful to impose the restriction that it should not be used if some variable with free occurrences in N should have one of those occurrences bound in $M[x{:=}N]$. (In addition, we must now understand $M[x{:=}N]$ to mean the result of replacing each *free* occurrence of x in M by N.) In order to cope with this contingency, Church instituted the α-step:

$$\lambda yM = \lambda z(M[y{:=}z]) \tag{1.10}$$

Note: To avoid confusion of free and bound variables in (1.10), we must have two restrictions: there are no free occurrences of z in M, and no occurrence of z in $M[y{:=}z]$ that resulted from replacing an occurrence of y in M by z is bound. Now we have

$$\oplus y = (\lambda x(\lambda z(x + z)))y = \lambda z(y + z) \tag{1.11}$$

We get the intermediate formula from the left one by an α-step. So we have

$$\oplus yx = (\lambda z(y + z))x = y + x \tag{1.12}$$

Equation (1.12) is just what $\oplus$ is supposed to do.

If we can get from M to N by a succession of steps, possibly null—each of which is either an α-step or a β-reduction or a β-expansion—we say that M is convertible to N; we write M conv N. If M conv N by a succession of steps, none of which is a β-expansion, we say that M is reducible to N; we write M red N.

John McCarthy worked several ideas of the LC into LISP. He clearly recognized procedures as functions of one argument. In the LC, such functions can be applied to each other and give such functions when applied. In LISP, it is possible to apply one procedure to another, and on occasion get another procedure.

As we said earlier, the K and S, and things built exclusively of them, are called combinators. Can we commingle combinators and lambda-expressions? Yes, indeed, with no trouble whatever. Moreover, variables, or indeterminates, may be freely included.

Note that if we replaced S by $\lambda x(\lambda y(\lambda z(xz(yz))))$ in (1.2), we could still conclude that (1.2) holds by three β-reductions. The lambda-expression just given would usually be abbreviated to $\lambda xyz(xz(yz))$. So we can define S by a lambda-expression. Church had decided that λxM should be formed only when there are free occurrences of x in M, and thus he could not get a lambda-expression to correspond to K. It could be done if we relaxed the requirement that there be at least one free occurrence of x in M to form λxM. Hence the LC, as originally set up by Church, seems a trifle weaker than the combinatory calculi of Schönfinkel and Curry. For present-day applications, either would serve perfectly well (this takes some proving) so that the difference is just something to niggle over—quite insignificant. Originally, it was not known that the difference was slight, and Rosser (1935) invented a couple of other combinators, in place of K and S, with which he set up an exact equivalent of the LC. Like Curry's, his system had the extensional property and a weak form of consistency; hence the LC also has these attributes.

The LC (and hence the combinatory calculi) has a fixed-point theorem. Given a function F, one can find a Φ such that

$$F\Phi = \Phi \tag{1.13}$$

Proof Take

$$\Phi = \phi\phi, \text{ where } \phi = \lambda xF(xx)$$

There is an obvious functional relationship between Φ and F, namely

$$\Phi = YF \tag{1.14}$$

where

$$Y = \lambda f((\lambda x f(xx))(\lambda x f(xx))) \tag{1.15}$$

Curry and Feys (1958, pp. 177–179) call Y the paradoxical combinator. The property

$$F(YF) = YF \tag{1.16}$$

for each F is noted, which the authors thought to be paradoxical when they wrote it. The property (1.16) makes Y useful in some of the modern treatments of combinators (Turner 1979*a*, p. 37).

2. A Debacle

The LC and the combinatory calculi were fairly promptly embedded in systems that had some of the earlier attributes of logical systems (Church 1932; Curry 1934*a*). The results turned out to be inconsistent. This circumstance was first proved in Kleene and Rosser (1935) by a variation of the Richard paradox. Later, Curry (1942) got a simpler proof, related to the Russell paradox, with the following simple form. Suppose we have the two familiar logical principles:

$$P \supset P \tag{2.1}$$

$$(\mathrm{P} \supset (\mathrm{P} \supset \mathrm{Q})) \supset (\mathrm{P} \supset \mathrm{Q}) \tag{2.2}$$

together with modus ponens (if P and $P \supset Q$, then Q). We undertake to prove an arbitrary proposition A. We construct a Φ such that

$$\Phi = \Phi \supset A \tag{2.3}$$

To do this, we take $F = \lambda x(x \supset A)$ in the fixed-point theorem. By (2.1), we get

$$\Phi \supset \Phi$$

Applying (2.3) to the second Φ gives

$$\Phi \supset (\Phi \supset A)$$

By (2.2) and modus ponens, we get

$$\Phi \supset A$$

By (2.3) reversed, we get

$$\Phi$$

By modus ponens and the last two formulas, we get

$$A$$

This debacle is usually referred to as the Curry paradox, by analogy with the Russell paradox.

F. B. Fitch (1936; 1952) proposed to avoid the Curry paradox by weakening the LC (or equivalent combinatory calculus) so that the fixed-point theorem fails. He also weakened modus ponens a bit. He has proved consistency for his system, but it is much too weak to be considered as a foundation for mathematics. He and his students have continued intermittently to the present to come out with improvements, but the system is still extremely weak.

W. Ackermann (1950; 1952; 1953) proposed keeping the full-strength LC, but crippling implication badly. He proved the consistency of the system, but it was hopelessly weak as a foundation for mathematics, and I know of no recent interest in it.

Curry (1934*b*; 1936) kept the full-strength combinatory calculus, but added a fragmentary theory of types and a weakened version of implication. He introduced a notion of functionality F, such that if Z is a function and X and Y are types, then $FXYZ$ is to denote that if U is of type X, ZU is of type Y. It turned out that the "natural" axioms for functionality lead to a contradiction (Curry 1955); by imposing suitable restrictions, however, all is well (Curry 1956). Since then, Curry and his students have made extensive developments. Two major works (Curry and Feys 1958; Curry, Hindley, and Seldin 1972) are landmarks. Even so, adoption of the system as a foundation for mathematics has not progressed at all, though the system has some capability in that direction. A first attempt to show this appeared in Cogan (1955). Unfortunately, the particular dialect of combinatory logic used by Cogan turned out to be inconsistent (see Titgemeyer 1961; Bunder 1974). Bunder was able to reassert the capability and gives an up-to-date discussion (1980).

3. Where Do We Go From Here?

A number of people are continuing to develop the systems of Fitch and Curry, but there is no likelihood that either will be adopted as a foundation for mathematics. It was originally expected that the LC or a combinatory calculus should be a part of some system used as a foundation for mathematics. Surprisingly, the LC (or a combinatory calculus) has turned out to be of importance in its own right. So we have to ask the questions that are asked about any logical system.

1. What about consistency?
2. What about completeness?
3. What about models?
4. What about the connection with computers?

At the time the LC and the combinatory calculi were being developed, no one asked the fourth question. Computers had not yet been invented!

4. What About Consistency?

We observed earlier that the LC and the combinatory calculi have a weak consistency, in that one cannot prove that all functions are equal to each other. Fairly early (see Church and Rosser 1936; reproduced in Church 1941) a considerably stronger form of consistency was proved for the LC, embodied in the Church-Rosser Theorem (referred to hereafter as C-R-T), namely:

Suppose X_0 red X_1 and X_0 red X_2. Then there is an X_3 such that both X_1 red X_3 and X_2 red X_3.

A lambda-formula is said to be in normal form if it has no part on which we can perform a β-reduction. A lambda-formula X is said to have a normal form Y if Y is in normal form and X conv Y. We can prove the following theorem.

If X has a normal form Y, then X red Y and Y is unique; except possibly for a few cosmetic α-steps.

We prove the first part of this by induction on the number of operations in X conv Y. The idea is as follows. Suppose we go from X to W_1 by a β-expansion, next from W_1 to W_2 by a β-reduction, and finally from W_2 to Y by a second β-reduction. Then W_1 red X and W_1 red Y. By C-R-T, there is a W such that X red W and Y red W. But Y is in normal form, so that in the reduction from Y to W there cannot be any β-reductions—only α-steps. Thus, except for cosmetic uses of α-steps, Y is W, and we have X red W. For uniqueness, suppose Z is another normal form of X. It is also a normal form of Y, and Y red Z. But Y is in normal form, so the reduction from Y to Z can consist only of α-steps.

The lambda-formulas of interest mostly have normal forms. These normal forms constitute a foundation, on which is erected an elaborate superstructure. Each normal form has its own individual superstructure, however, not overlapping the superstructures of the other normal forms.

Because formulas of the LC can be identified with formulas of a combinatory calculus and vice versa, there are superstructures in the combinatory calculus corresponding to those of the LC.

In (1.1) and (1.2), we can consider going from left to right as a reduction. We can look for parallels to C-R-T, we can define normal forms, etc. There was investigation of these questions.

The original proof of C-R-T was fairly long and very complicated. M. H. A. Newman (1942) made the point that the proof was basically topological. He generalized the universe of discourse and defined a relation with properties similar to a β-reduction. He proved a result similar to C-R-T by topological arguments. Curry (1952) generalized the Newman result, with the intention that it would be relevant to similar considerations in the combinatory calculi. Unfortunately, it turned out that neither the Newman result nor the Curry generalization entailed C-R-T in the intended systems because the systems did not satisfy the hypotheses of the key theorems. This was discovered by David E. Schroer, whose counterexample is recorded in Rosser (1956). In Schroer (1965) still further generalizations of the Newman and Curry results are derived, which indeed do entail C-R-T in assorted systems. (The Schroer paper is 627 typed pages; this hardly contributes to the cause of shorter and simpler proofs of C-R-T.)

Chapter 4 of Curry and Feys (1958) is devoted to a proof of C-R-T for the LC and to related matters; it is not recommended for light reading. Hindley (1969; 1974) discusses proofs of C-R-T for the LC and closely related systems.

These various proofs all stemmed generally from the Newman approach, with an emphasis on the topological structure. However, lambda-formulas and combinators have a marked, though specialized, tree structure. G. Mitschke (1973) used the tree properties a bit in deriving a proof of C-R-T, and B. K. Rosen (1973) went much further. He worked with general trees and relationships among them. Because lots of things have a tree structure, his results have applications beyond proving C-R-T. He applies his results to the extended McCarthy calculus for recursive definition (see McCarthy 1960) and verifies a conjecture in Morris (1968). He also applies his results to tree transducers in syntax-directed compiling. With all that, the proof of C-R-T did not come easy. He had to prove C-R-T's for several related systems, and then derive the C-R-T for the LC by some trickery.

Meanwhile, a genuine simplification for the proof of C-R-T had come in sight: Martin-Löf (1972). It is agreed that Martin-Löf got some of his ideas from lectures by William Tait. An exposition of the proof of C-R-T according to Tait and Martin-Löf appears as Appendix 1 in Hindley, Lercher, and Seldin (1972). A shorter exposition appears in Barendregt (1981, pp. 59–62). We will give what seems to us a still shorter and more perspicuous proof of C-R-T.

What seems to be the main difficulty of the proof? Let us look at the minimal case. Suppose X_0 has two parts, $(\lambda x\, W_1)\, V_1$ and $(\lambda x\, W_2)\, V_2$. Let X_0 red X_i by performing a β-reduction on $(\lambda x\, W_i)\, V_i$, for $i = 1,2$. If $(\lambda x\, W_1)\, V_1$ and $(\lambda x\, W_2)\, V_2$ reside in totally disjoint parts of X_0, there is no trouble. To get X_3 we perform a β-reduction on the $(\lambda x\, W_2)\, V_2$ that still resides in X_1 and on the $(\lambda x\, W_1)\, V_1$ that still resides in X_2.

Note that the reductions from X_1 to X_3 and from X_2 to X_3 each use exactly one β-reduction.

Suppose that $(\lambda x\, W_2)\, V_2$ is part of V_1. X_2 will contain $(\lambda x\, W_1)\, V_3$, where V_3 is the result of a β-reduction of $(\lambda x\, W_2)\, V_2$ inside V_1. As a candidate for X_3, we perform a β-reduction on the $(\lambda x W_1) V_3$ of X_2. How about getting from X_1 to X_3? Where X_0 had $(\lambda x W_1) V_1$, X_1 will have $W_1[x:=V_1]$. If there had been only one free occurrence of x in W_1, then $W_1[x:=V_1]$ will contain a corresponding V_1; we change this to V_3 by a β-reduction on $(\lambda_2\, W_2)\, V_2$, and we have arrived at X_3. But W_1 may well contain several free occurrences of x. Then $W_1[x:=V_1]$ will contain several V_1's. We can change them one after another to V_3's, which will result in X_3. But there is no way we can get from X_1 to X_3 by a single β-reduction.

In the language of Barendregt (1981, p. 54), β-reduction does not have the diamond property.

The difficulty is that it may take several β-reductions to get from X_1 to X_3. This should have suggested working with a string of β-reductions instead of only one. Why it took more than 30 years for this to occur to anyone is a mystery.

If we call a β-reduction or α-step a step, then a string of them will be a walk. But we cannot allow just any old string. What we are aiming for is that if X_0 walk X_1 and X_0 walk X_2, there is an X_3 such that X_1 walk X_3 and X_2 walk X_3. If we put the right restrictions on the steps allowed in a walk, we can do this.

We frame our restrictions for a walk as follows.

1. A walk may contain no steps at all.
2. It may contain α-steps at will.
3. If a number of parts $(\lambda x W_i) V_i$ fail to overlap at all, the corresponding β-reductions may be done in any order.
4. Let $(\lambda x W) V$ be reduced to $W[x:=V]$ in a β-reduction of the walk. Inside that part, $W[x:=V]$, no subsequent β-reductions may be performed in the walk, and likewise no β-reduction of all of $W[x:=V]$, in case it has the requisite structure (which it could). However, α-steps may be performed inside $W[x:=V]$.

The relation $\xrightarrow[1]{}$ in Barendregt (1981, p. 60) is likely closely related to our notion of a walk, but it is not exactly the same. For the key lemma, Barendregt uses something like induction on the number of steps from X_0 to X_1 whereas we use induction on the number of symbols in X_0. This makes quite a difference.

We need a lemma, which is about as follows.

Suppose X walk Y. Then $X[x:=P]$ walk $Y[x:=P]$ by a completely analogous series of β-reductions.

To see this, note that α-steps do nothing to the free occurrences of x. A β-reduction can rearrange the free occurrences of x. It can even replicate them, as would happen if the β-reduction were from $(\lambda y S) T$ to $S[y:=T]$; if there are several free occurrences of y in S, they would each be replaced by T, and any free occurrences of x in T would be thereby replicated.

If occurrences of P are put for the free occurrences of x in X, a completely analogous series of β-reductions is possible, and all it will do is rearrange or replicate the P's just as the walk from X to Y did for the free occurrences of x. At the end, we have $Y[x:=P]$ as the result. The fact that the restrictions for a walk were satisfied in going from X to Y assures us that they will be satisfied in going from $X[x:=P]$ to $Y[x:=P]$.

Actually, the lemma is not quite true, because of the possibility of confusion of free and bound variables. Before trying the first step from $X[x:=P]$ to $Y[x:=P]$, we could already be in trouble if some of the free variables in P became bound when P was put for x in X. A very close relative of the lemma, sufficient for our purposes, is true, however.

Lemma. Suppose X walk Y. In X, change all bound variables by α-steps to a set of distinct variables that have no occurrences in X or P. This gives X^1, for which there is a Y^1 such that X^1 walk Y^1 by essentially the same β-reductions as were used for X walk Y. Then $X^1[x:=P]$ walk $Y^1[x:=P]$ by a completely analogous series of β-reductions.

We first note that there will be no need for α-steps in either the walk from X^1 to Y^1 or the walk from $X^1[x:=P]$ to $Y^1[x:=P]$. All possibility of confusion of bound variables has been sidestepped in changing from X to X^1, and we can now use the argument given originally.

Note that this lemma is very nearly the same as proposition 2.1.17(i) in Barendregt (1981, p. 28). It is also closely related to proposition 3.1.16 (p. 55). In Barendregt's terminology, our lemma says that a walk is substitutive.

Diamond Property

If X_0 walk X_1 and X_0 walk X_2, there is an X_3 such that X_1 walk X_3 and X_2 walk X_3.

In other words, there is an X_3 that is the fourth vertex of a diamond, with a walk along each edge. See Figure 1 (ahead), where W_4 occupies a fourth vertex to correspond to X_0, W_1, and W_3 on the three upper vertices; then W_5 occupies a fourth vertex to correspond to W_1, W_2, and W_4 on the three upper vertices; and so on.

The following is a proof by induction on the number of symbols in X_0.

Case 1. If X_0 has a single symbol, the diamond property is immediate.

Case 2. Let X_0 be $\lambda x M_0$. Replace all bound variables in X_0 by distinct variables $z_1, z_2, \ldots, z_n$ not occurring in X_0. Hereby X_0 is changed to $\lambda z_1 M_0^1$. X_i will be $\lambda x_i M_i$, for $i = 1,2$. The β-reductions that carried X_0 to

X_i will also carry $\lambda z_1 M_0^1$ to $\lambda z_1 M_i^1$, for $i = 1,2$ and without involving z_1. Clearly we can go from X_i, namely $\lambda x_i M_i$, to $\lambda z_1 M_i^1$, for $i = 1,2$ merely by using α-steps to change assorted bound variables to z_1, z_2, ..., z_n. Since the walks from $\lambda z_1 M_0^1$ to $\lambda z_1 M_i^1$ do not involve z_1, there are walks from M_0^1 to M_i^1, for $i = 1,2$. M_0^1 has fewer symbols than X_0. Consequently, by the hypothesis of our induction, there must be an M_3 with walks from M_i^1 to M_3, for $i = 1,2$. Hence there are walks from $\lambda z_1 M_i^1$ to $\lambda z_1 M_3$, for $i = 1,2$. Take X_3 to be $\lambda z_1 M_3$. As noted earlier, we go from X_i to $\lambda z_1 M_i^1$ by α-steps alone, for $i = 1,2$. Preface these to the walks from $\lambda z_1 M_i^1$ to $\lambda z_1 M_3$, and we get walks from X_i to X_3, for $i = 1,2$.

Case 3. Let X_0 be $M_0 N_0$.

Subcase 1. X_i is $M_i N_i$ with M_0 walk M_i and N_0 walk N_i, all for $i = 1,2$. Then there are M_3 and N_3 with M_i walk M_3 and N_i walk N_3, both for $i = 1,2$. Take $X_3 = M_3 N_3$.

Subcase 2. M_0 is $\lambda x W_0$. X_1 is $(\lambda x_1 W_1) N_1$, where $\lambda x W_0$ walk $\lambda x_1 W_1$ and N_0 walk N_1. In getting from X_0 to X_2 there has been a β-reduction on the "descendant" of x in $\lambda x W_0$. By restriction 4, this β-reduction had to be the last β-reduction in the walk to X_2. Hence the last formula before this reduction had to be $(\lambda x_2 W_2) N_2$, which the reduction took to $W_2[x_2:=N_2]$. This formula is therefore X_2, except for possible α-steps following the last β-reduction. Consequently, $\lambda x W_0$ walk $\lambda x_2 W_2$ and N_0 walk N_2.

Replace all bound variables in X_0 by distinct variables z_1, z_2, ..., z_n not occurring in X_0. Hereby X_0 is changed to $(\lambda z_1 W_0^1)\ N_0^1$. The β-reductions that carried $\lambda x W_0$ to $\lambda x_i W_i$ and N_0 to N_i will also carry $\lambda z_1 W_0^1$ to $\lambda z_1 W_i^1$ and N_0^1 to N_i^1, all for $i = 1,2$; see the proof in Case 2 earlier. We can go from $\lambda x_i W_i$ to $\lambda z_1 W_i^1$ and from N_i to N_i^1, both for $i = 1,2$, merely by using α-steps to change assorted bound variables to z_1, z_2, ..., z_n.

By familiar reasoning, we see that W_0^1 walk W_i^1, for $i = 1,2$. By the hypothesis of our induction, there are W_3 and N_3 such that W_i^1 walk W_3, and N_i^1 walk N_3, both for $i = 1,2$. Hereby we infer $(\lambda z_1 W_i^1) N_i^1$ walk $(\lambda z_1 W_3) N_3$, for $i = 1,2$. We take X_3 to be $W_3[z_1:=N_3]$. It follows readily that X_1 walk X_3. As $\lambda x_2 W_2$ and N_2 go to $\lambda z_1 W_2^1$ and N_2^1 by α-steps, we can go from X_2, which is $W_2[x_2:=N_2]$ to $W_2^1[z_1:=N_2^1]$ by α-steps. As N_2^1 walk N_3, we can get from $W_2^1[z_1:=N_2^1]$ to $W_2^1[z_1:=N_3]$ by a walk consisting of walks on the nonoverlapping N_2^1's. Since we have a walk from W_2^1 to W_3, our lemma tells us that there is a walk from $W_2^1[z_1:=N_3]$ to $W_3[z_1:=N_3]$, which is X_3. Now the β-reductions we took in going from X_2 to $W_2^1[z_1:=N_3]$ were all on N_2^1's that had been put for z_1's in W_2^1. None of them can violate restriction 4 as we walk on down to $W_3[z_1:=N_3]$ by our lemma from $W_2^1[z_1:=N_3]$. We can therefore attach this walk to the one from X_2 to $W_2^1[z_1:=N_3]$ to conclude that there is a walk from X_2 to $W_3[z_1:=N_3]$, which is X_3.

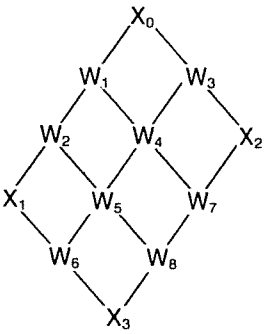

Figure 1. Diamond Property.

Subcase 3. Subcase 3 is like Subcase 2, except X_1 and X_2 are interchanged. We make suitable interchanges in the proof of Subcase 2.

Subcase 4. M_0 is $\lambda x W_0$. In getting from X_0 to X_i, for $i = 1,2$, there has been a β-reduction on the "descendant" of x in $\lambda x W_0$. By restriction 4, this β-reduction had to be the last β-reduction in the walk to X_i, for $i = 1,2$. Hence the last formula before this reduction had to be $(\lambda x_i W_i) N_i$, for $i = 1,2$, which the reduction took to $W_i[x_i:=N_i]$. This formula is therefore X_i, for $i = 1,2$, except for possible α-steps following the last β-reduction. Consequently, $\lambda x W_0$ walk $\lambda x_i W_i$ and N_0 walk N_i, both for $i = 1,2$.

We now proceed exactly as in Subcase 2, except that both X_1 and X_2 are handled in the way we handled X_2 in Subcase 2. We disregard the treatment of X_1 as presented in Subcase 2.

Now we prove something that looks like C-R-T.

If X_0 goes to X_i by a succession of walks, for $i = 1,2$, there is an X_3 such that X_i goes to X_3 by a succession of walks, for $i = 1,2$.

The proof is so easy that if we carry out the details for a special case, the whole thing becomes obvious. Let X_0 walk W_1 walk W_2 walk X_1 and X_0 walk W_3 walk X_2. By the Diamond Property, we can fill in W_i's and X_3 to be corners in Figure 1.

Because each β-reduction or α-step taken alone is a walk, C-R-T follows by the previous result.

Although the proof in Newman (1942) failed to prove the C-R-T for the LC, it does prove a C-R-T for a fairly general universe of discourse. Some cases of this have been found to be useful, although they may not have much in common with the LC; see Book (1982) and several of the authors cited in bibliographic references in Book (1982).

5. What About Completeness?

At first sight, it appears that the LC is so weak that it is absurd even to raise the question. As indicated in Church (1933) and amplified in Kleene (1935), however, the positive integers can be defined in the LC. If n is a positive integer, we let

$$\lambda f(\lambda x(f(f(\ldots(f(fx))\ldots))))$$

denote the integer n, where there are n f's inside parentheses. This makes one form of recursive definition easy. If $F(n)$ is to be defined by

$$F(1) = GA \tag{5.1}$$

$$F(n + 1) = G(Fn) \tag{5.2}$$

we can take F to be

$$\lambda n(nGA) \tag{5.3}$$

With this definition,

$$F1 \text{ red } GA$$

$$F2 \text{ red } G(GA)$$

$$F3 \text{ red } G(G(GA))$$

etc.

There is no zero in this system. We would prefer the recursive definition to be given by

$$F(1) = A \tag{5.4}$$

$$F(n + 1) = G(Fn) \tag{5.5}$$

Stephen C. Kleene (1935) worked out a way to do this and opened the door to still more difficult recursive definitions. A crucial step was Kleene's discovery (see Kleene 1934) that the connection between the LC and the combinatory calculus established by Rosser (1935) (but known to Kleene in the fall of 1932) makes possible definition by cases in the LC under the most general circumstances.

Kleene discovered more and more definitions of functions from integers to integers. Some never-published investigations by Rosser disclosed so many more that in the fall of 1933, based on Rosser's work and results of Kleene that Church had seen (later published in Kleene 1935), Church speculated that every effectively calculable function from positive integers to positive integers is definable in the LC. It was known (Church and Rosser 1936) that every function from positive integers to positive integers that is definable in the LC is effectively calculable. Finally, in early 1934, Church gave unequivocal support to what is now known as "Church's Thesis" (Church 1935; 1936).

Church's Thesis

Effectively calculable functions from positive integers to positive integers are just those definable in the LC.

Because "effectively calculable" is an intuitive notion, Church's Thesis is not susceptible of proof, but it does state a strong—and quite unexpected—version of completeness.

In lectures in 1934, Kurt Gödel gave a definition of "general recursive function" (see Davis 1965, pp. 69–71). He refrained from espousing it as a criterion of effective calculability until after the results cited next (including Turing's work) had appeared. Kleene (1936) showed that general recursiveness is the same as being definable in the LC, which lent strong support to Church's Thesis.

Independently, Alan M. Turing had been developing the abstract idea of a computer, the so-called Turing machine (Turing 1936). He thought that "effectively calculable" should be taken to be the same as calculable on a Turing machine, but he proved (1937) that that is the same as being definable in the LC. This result explains why the lambda-calculus and the combinatory calculi can (and do) play such an important role in the theory of computer programming, and such matters.

Also independently, Emil L. Post (1936) had developed ideas similar to those of Turing (Turing published first, by a few months). Later, Post (1943) proposed still another definition of "effectively calculable," which turned out to be equivalent to those already given. A. A. Markov (1951; 1961) gave yet another definition, which was also proved to be equivalent, as was a definition by R. M. Smullyan (1961) using his "elementary formal systems."

With the development of actual computers, which are finite approximations for a universal Turing machine, interest in all these matters has been much intensified. Kleene and Vesley (1965, p. 3) list 150 contributions to the subject by October 15, 1963. By now there are far more.

Kleene's early developments in recursion theory were of much importance for computing. Now, although he still uses many notations from the LC, he has diverged far from it into an area that is essentially of no use in computing—even though active and of interest to many people.

I think it might be of interest for me to recount, from a personal view, some of the matters involved in the development of Church's Thesis, general recursivity, and its relation to Church's Thesis. I am grateful to Kleene for reading this, and for refreshing my memory at spots, especially those having particularly to do with Kleene himself.

The developments that led to conjecturing the connection between λ-definability and effective calculability were initiated in lectures by Church in the fall semester of 1931–1932, which Kleene and I attended (see Church 1933). Church gave the definition of positive integer expounded earlier, and noted that the recursive definition set out in (5.1) and (5.2) could be defined in LC.

In February, immediately after the end of the lectures, Kleene astonished everyone by giving a definition of the predecessor function in LC.

For the next year and a half, Kleene continued to find ways to define various kinds of functions in the LC. The key details appear in Kleene (1935), which was part of his Ph.D. thesis (presented to Church in June 1933). I had many discussions about these matters with Kleene and was quite familiar with the details. From Kleene's techniques, Church's Thesis should have been obvious in June 1933 to Church, Kleene, and me. It had apparently not entered our minds to consider it yet, and none of us suggested it.

Perhaps I was not as convinced by Kleene's techniques as I should have been. In the fall of 1933 I kept searching out subtle effectively calculable functions, and attempting to find functions in the LC to represent them. Naturally, I succeeded. Kleene had gone to his mother's farm in Maine for about seven months in July 1933, so I did not disclose any of these investigations to him. Gödel was at the Institute for Advanced Study, but apparently had no interest in the LC (at that point). Only Church heard of my endeavors. One time, in late 1933, I was telling him about my latest function in the LC. He remarked that perhaps every effectively calculable function from positive integers to positive integers is definable in LC. He did not say it with any firm conviction. Indeed, I had the impression that it had just come into his mind from hearing about my latest function. With the results of Kleene's thesis and the investigations I had been making that fall, I did not see how Church's suggestion could possibly fail to be true. In fact, I immediately berated myself (silently) for not having seen the obvious a month or two before, so that I would have made that proposition to Church before he made it to me.

The question had only shortly before come up whether Church's system of logic was consistent (it was not). This question seemed much more overriding and engaged most of our attention for the rest of the semester. Indeed, I doubt if either of us again considered the connection between effective calculability and λ-definability until after Christmas. By that time, it had become fairly conclusive that Church's system was inconsistent, and our thoughts turned to other matters. After Kleene returned to Princeton on February 7, 1934, Church looked more closely at the relation between λ-definability and effective calculability. Soon he decided they were equivalent, and Kleene named the proposition "Church's Thesis." Church got the thesis into print as soon as possible (1935; 1936).

Kleene recounts his reaction to Church's proposal in his *Annals* article (Kleene 1981, p. 59). Kleene's attempt at a disproof failed, but the attempt was so illuminating that he "became overnight a supporter of the thesis."

At about the same time, Church proposed Church's Thesis to Gödel. As noted earlier, Gödel knew little about the LC and rejected the idea (see Kleene 1981, p. 59). Church challenged Gödel to come up with at least a partially satisfactory definition of effective calculability. This apparently stimulated Gödel to refine an earlier suggestion by Herbrand and produce the definition of general recursiveness that he presented in lectures later that spring, cited earlier. His definition is the one now in general use, although Kleene published an equivalent rephrasing (1936).

Church, Kleene, and I each thought that general recursivity seemed to embody the idea of effective calculability, and so each wished to show it equivalent to λ-calculability. Church (1936) has commented about what transpired then (reproduced in Davis 1965, pp. 90, 99). Church, Kleene, and I each had pretty good drafts of a proof ready before the end of the semester, but by somewhat different methods. There was enough discussion and interchange of information during this period that we became convinced of the equivalence of general recursivity and λ-calculability. Church (1936, footnote 3) gives credit to Kleene for getting the full proof first, but gives me priority for a proof of a similar result. Footnote 3 (in Davis 1965, pp. 90, 99) has to be taken as saying that Church and I made considerable progress to a proof, even though Kleene got there first, and not that Kleene's proof (as given in Kleene 1936) owes anything to Church or me. There was much discussion and interchange of ideas at the time, but Kleene's recollection is that this did not particularly help him to find his proof. Certainly, in the proof itself, all difficult points are handled by citations of results that Kleene had obtained for various reasons.

With the equivalence of general recursiveness and λ-definability established, Church, Kleene, and I expected that Gödel would join us in supporting Church's Thesis. Gödel still had reservations, however, and it was not until at least two or three years later, after Turing's investigations, that Gödel finally became a believer (as did Turing also).

There have been some objections to Church's Thesis. Moschovakis (1968) gives a simultaneous review of four papers, by Jean Porte, László Kalmár, Rózsa Péter, and Elliott Mendelson. The first three papers attempt in various ways to discredit Church's Thesis. The paper by Mendelson discusses the three papers and undertakes to show that their criticisms are ill-founded. In the opinion of the reviewer, he succeeds quite adequately. I know of no recent attacks on Church's Thesis, and it seems to be generally accepted as an important, if unorthodox, version of completeness for the LC.

6. What About Models?

A classic theorem says that if a logic is consistent, it will have a model—indeed a denumerable one. The LC is so different in structure from the usual logics that the theorem does not apply to it.

Why do we wish a model? If we have a framework with a lot of structure, and the logic is isomorphic to some part of the framework, the structure in the framework can contribute to our understanding of the logic. We can always manufacture a superficial model by taking equivalence classes of objects in the logic. The only structure this model has is forced by the logic. No additional understanding can come from studying the structure of the model, and such a model does little good.

For a long time, this was the only kind of model that was found for the LC. Finally, with encouragement from Christopher Strachey, Dana Scott (1970) hit on a way of making some really useful models. They could be constructed either in the category of topological spaces or in the category of lattices. Barendregt (1981) gives a model similar to the Scott one, but in a still more general framework—namely, the category of complete partial orders. In one sense, this is good because we can derive still more properties of the LC in this general category. Suppose, however, that we would like just to see a model without having to learn all the algebra involved in topological spaces, lattices, or complete partial orders. Some people have been working in that direction—to get a model without all the algebraic baggage. (Most references are unpublished; see Plotkin 1972, Engeler 1979, and Meyer 1982.) Albert R. Meyer (1982) gives a fairly complete and coherent account. He says the model originated with Plotkin, was improved by Engeler, and was further improved by Meyer himself. My account is taken from the Meyer paper.

Start with a nonempty set A; the unit class consisting of the ordered pair $\langle \phi, \phi \rangle$ will do, where ϕ is the null class. Enlarge A to the least set B containing A and all ordered pairs $\langle \beta, b \rangle$, where β is a finite subset of B and b is in B.

The model consists of all subsets of B. For two members, C and D, of the model, define

$$(CD) = \{b \varepsilon B \mid \langle \beta, b \rangle \varepsilon C \text{ and } \beta \subseteq D\} \qquad (6.1)$$

To show that this contains a model of the combinatory calculus, we identify two elements K and S:

$$K = \{\langle \alpha, \langle \beta, b \rangle \rangle \mid b \varepsilon \alpha \text{ and } \alpha, \beta \text{ finite subsets of } B\} \qquad (6.2)$$

$$S = \{\langle \alpha, \langle \beta, \langle \gamma, b \rangle \rangle \rangle \mid b \varepsilon \alpha \gamma (\beta \gamma) \text{ and } \alpha, \beta, \gamma \text{ finite subsets of } B\} \qquad (6.3)$$

We can verify fairly easily that

$$KCD = C \qquad (6.4)$$

$$SCDE = CE(DE) \qquad (6.5)$$

for all elements of the model. A close relative of the extensional property holds (see Meyer 1982).

Because the LC is so closely related to the combinatory calculi, it is not surprising that something similar can be put together as a model for the LC. Meyer (1982) has full details.

7. What About the Connection with Computers?

The computer connection proceeds in two directions. We can use computers to manipulate combinators or formulas of the LC, or we can use properties of combinators and the LC to help in programming or to develop ideas of use for computers.

Looking to the first, an obvious approach would be to represent the combinators, or formulas of the LC, as lists or arrays in the computer memory. In fact, these formulas are tree structures, and might better be so represented on the computer. Knowing the location of only the root of the tree suffices to reconstruct the entire tree. So the trees (entire formulas) can be identified by single memory locations, instead of by elaborate diagrams or linearizations of diagrams.

The idea is simple. Suppose A and B are combinators, and we have put their roots at memory locations a and b.Then we represent $C = (AB)$ by locating its root at memory location c; in c we put the ordered pair of numbers a and b. If we wish to know the structure of C, we are told to look at location c. There we find $\langle a, b \rangle$, which tells us that C has the form (AB), and that to know the form of A we should look in location a, and similarly for B.

Besides the convenience in referring to a formula, this method of representation allows economies of memory which are not possible when a formula is

represented by a list. For an extreme example, suppose $B = ((AA)(AA))$, where A requires 1000 memory locations for its representation. To represent B as a list would require four repetitions of the listing of A, together with attendant parentheses—a total of 4006 locations. With the tree representation, let A have its root at a; we may still suppose that the entire representation of A fills 1000 locations. At some convenient memory location d, we put $\langle a, a \rangle$, which denotes $D = (AA)$. At another empty memory location b, we put $\langle d, d \rangle$, which denotes (DD). But (DD) is $((AA)(AA)) = B$. Thus, with A represented in 1000 memory locations, we require only 1002 locations to represent $((AA)(AA))$.

Another advantage of the tree representation is that it lends itself to what is called "lazy evaluation." Suppose a part M occurs several times in a formula X. If X is represented as a list, the several occurrences of M are each written out in full. Unless extraordinary measures are taken, each of the occurrences of M will be evaluated separately, and independently, in the course of evaluating X. With a tree structure, however, M will occur only once, but with various pointers "pointing" to it. Hence it will be evaluated only once.

These and many related matters are taken up by George W. Petznick (1970). Consider a typical program on a computer—say, for computing an approximation to the square root (two integers: a mantissa and an exponent). If we input an approximation for a real number (a mantissa and an exponent), the program will generate and output an approximation for the square root. So the program defines a function. Naturally, it is a computable function. So (by one of the equivalences supporting Church's Thesis) this function must be expressible by means of a combinatory formula. If suitable hardware, or software simulations thereof, is available, the calculation can be done solely by combinatory manipulations.

Something of the sort had been proposed for lambda-formulas by P. J. Landin (1965; Steel 1966, pp. 266–294), who defined and used what he calls SECD machines. Here he was unavoidably involved in a difficult problem of handling the complicated substitutions properly. If he had used combinatory formulas instead, the problem would be much simplified. Also, Landin tried to superpose the lambda-formulas on top of the usual computer software, producing a greatly complicated assignment problem. If he had dispensed with the usual computer software, and worked only with combinatory formulas stored in the memory (preferably as trees), the assignment problem would simply have disappeared.

Petznick's thesis (1970) showed that it is possible to design a computer to work exclusively with combinatory formulas, stored as trees. There is no assignment problem, and application takes the place of substitution. Because application is the basis of the tree structure, it is handled automatically. The hardware one would have to build to handle this would be quite simple—or it could be handled with present hardware by a suitable software simulation.

Left to right in 1982: J. Barkley Rosser, Annetta Rosser, Haskell Curry, Virginia Curry, Alonzo Church.

Petznick's thesis managed to evade everybody's attention, and nothing more was done in that area for a while. After some years, work similar to Petznick's (and extending it) began to appear and has quickly blossomed. It now engages the attention of a considerable number of people, all of whom seem to be quite unaware of Petznick's work.

There is quite a ferment of activity just now, and several papers were presented at the 1982 ACM Symposium on LISP and Functional Programming at Pittsburgh; a set of proceedings is available (ACM order number 552820). It would surpass my powers as a soothsayer to determine what will develop as the key ideas; perhaps some have not yet emerged.

I will sketch a couple of trains of development, to give the reader some sort of idea what is happening. In so doing, I may fail to note something that will be of major importance, and so fail to give due credit to those who are working on it.

In Henderson and Morris (1976) appeared an idea for lazy evaluation. Turner (1979*a*; 1979*b*) carried this forward, and also showed how to condense combinatory formulas greatly, thereby alleviating what had been a problem for Petznick. More on that last point is given in Hughes (unp.). Two programs for manipulating combinators directly are CRS/1 (see References) and SKIM, which was announced in 1980 and is now being improved by a group at Cambridge Univer-

sity. Backus (1978) does not seem to be in the mainstream of this activity, but he has some quite novel combinatory functions, and something interesting may evolve out of his work.

It seems to be now established that operating directly on computers in combinatory format is not only feasible, but has some advantages. Even more useful results may be just around the corner. Or they may have already been announced without my appreciating their worth.

REFERENCES[1]

Ackermann, W. 1950. Widerspruchsfreier Aufbau der Logik I. Typenfreies System ohne Tertium non datur. *J. Symb. Logic 15*, 33–57.

Ackermann, W. 1952; 1953. Widerspruchsfreier Aufbau einer typenfreien Logik. I. *Math. Zeit. 55*, 364–384; II, *57*, 155–166.

Backus, J. 1978. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *CACM 21*, 613–641.

Barendregt, H. P. 1981. *The Lambda Calculus*. Amsterdam, North-Holland.

Book, R. V. 1982. Confluent and other types of Thue systems. *JACM 29*, 171–182.

Bunder, M. W. 1974. Some inconsistencies in illative combinatory logic. *Zeit. Math. Logik Grundlagen Math. 20*, 199–201.

Bunder, M. W. 1980. "The Naturalness of Illative Combinatory Logic as a Basis for Mathematics." In J. P. Seldin, and J. R. Hindley (eds.), to H. B. Curry, *Essays on Combinatory Logic, Lambda Calculus and Formalism*, New York, Academic Press, pp. 55–64.

Church, A. 1932. A set of postulates for the foundation of logic. *Annals of Math. 33*, 2nd series, 346–366.

Church, A. 1933. A set of postulates for the foundation of logic (second paper). *Annals of Math. 34*, 2nd series, 839–864.

Church, A. 1935. An abstract. *Bull. AMS 41*, 333.

Church, A. 1936. An unsolvable problem of elementary number theory. *Amer. J. Math. 58*, 345–363.

Church, A. 1941. "The Calculi of Lambda-Conversion." *Annals of Math. Studies 6*, Princeton, N.J., Princeton Univ. Press; 2nd ed., 1951.

Church, A., and J. B. Rosser. 1936. Some properties of conversion. *Trans. Amer. Math. Soc. 39*, 472–482.

Cogan, E. J. 1955. A formalization of the theory of sets from the point of view of combinatory logic. *Zeit. Math. Logik Grundlagen Math. 1*, 198–240.

CRS/1 specification, Beale Electronic Systems Ltd., Wraysbury, U.K.

Curry, H. B. 1929. An analysis of logical substitution. *Amer. J. Math. 51*, 363–384.

Curry, H. B. 1930. Grundlagen der kombinatorischen Logik. *Amer. J. Math. 52*, 509–536.

Curry, H. B. 1934*a*. Some properties of equality and implication in combinatory logic. *Annals of Math. 35*, 2nd series, 849–860.

Curry, H. B. 1934*b*. Functionality in combinatory logic. *Proc. Nat. Acad. Sci. 20*, 584–590.

Curry, H. B. 1936. First properties of functionality in combinatory logic. *Tôhoku Math. J. 41*, 371–401.

Curry, H. B. 1942. The inconsistency of certain formal logics. *J. Symb. Logic 7*, 115–117.

Curry, H. B. 1952. A new proof of the Church-Rosser theorem. *Nederl. Akad. Wetensch. 55*, Ser. A (*Indag. Math. 14*), 16–23.

Curry, H. B. 1955. The inconsistency of the full theory of combinatory functionality (Abstract). *J. Symb. Logic 20*, 91.

Curry, H. B. 1956. Consistency of the theory of functionality (Abstract). *J. Symb. Logic 21*, 110.

Curry, H. B., and R. Feys. 1958. *Combinatory Logic*. Amsterdam, North-Holland.

Curry, H. B., J. R. Hindley, and J. P. Seldin. 1972. *Combinatory Logic*. Vol. II, Amsterdam, North-Holland.

Davis, M. 1965. *The Undecidable*. New York, Raven Press.

Engeler, E. (ed.). 1971. Symposium on semantics of algorithmic languages. *Lecture Notes in Mathematics*, No. 188, New York, Springer-Verlag.

Engeler, E. 1979. Algebras and combinators. *Berichte des Inst. für Informatik*, No. 32, ETH Zurich, 12 pp.

Fitch, F. B. 1936. A system of formal logic without an analogue to the Curry W operator. *J. Symb. Logic 1*, 92–100.

Fitch, F. B. 1952. *Symbolic Logic, An Introduction*. New York, Ronald Press.

Henderson, P., and J. H. Morris. 1976. A lazy evaluator. *Proc. 3d ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, Atlanta, pp. 95–103.

Hindley, R. 1969. An abstract form of the Church-Rosser Theorem. I. *J. Symb. Logic 34*, 545–560.

Hindley, R. 1974. An abstract Church-Rosser Theorem. II: Applications. *J. Symb. Logic 39*, 1–21.

Hindley, J. R., B. Lercher, and J. P. Seldin. 1972. *Introduction to Combinatory Logic*. London Mathematical Society Lecture Note Series 7, Cambridge, Cambridge Univ. Press.

Hughes, R. J. M. Unp. "Graph-Reduction with Supercombinators." Oxford University Programming Research Group Technical Monograph PRG-28.

Kleene, S. C. 1934. Proof by cases in formal logic. *Annals of Math. 35*, 2nd series, 529–544.

Kleene, S. C. 1935. A theory of positive integers in formal logic. *Amer. J. Math. 57*, 153–173, 219–244.

Kleene, S. C. 1936. λ-definability and recursiveness. *Duke Math. J. 2*, 340–353.

Kleene, S. C. January 1981. Origins of recursive function theory. *Annals of the History of Computing 3*, 1, 52–67.

Kleene, S. C., and J. B. Rosser. 1935. The inconsistency of certain formal logics. *Annals of Math. 36*, 2nd series, 630–636.

Kleene, S. C., and R. E. Vesley. 1965. *The Foundations of Intuitionistic Mathematics*. Amsterdam, North-Holland.

Landin, P. J. 1965. A correspondence between ALGOL 60 and Church's lambda-notation. *CACM 8*, 89–101, 158–165.

Markov, A. A. 1951. Teoriya algorifmov. *Trudy Mat. Inst. Steklov 38*, 176–189. Trans.: "Theory of algorithms," No. OTS 60-51085, U.S. Department of Commerce, Office of Technical Services, 1961.

[1] Copies of some unpublished theses referenced here can be obtained from University Microfilms International, 300 N. Zeeb Road, Ann Arbor, MI 48106.

Martin-Löf, P. 1972. "An intuitionistic theory of types." University of Stockholm, manuscript.

McCarthy, J. 1960. Recursive functions of symbolic expressions and their computation by machine. *CACM 3*, 184–195.

Meyer, A. R. 1982. What is a model of the lambda calculus? *Information and Control 52*, 87–122.

Mitschke, G. 1973. Ein algebraischer Beweis für das Church-Rosser Theorem. *Archiv. für mathematische Logik 15*, 146–157.

Morris, J. H., Jr. 1968. "Lambda-Calculus Models of Programming Languages." MAC-TR-57, MIT Project MAC, Cambridge.

Moschovakis, Y. N. 1968. A review. *J. Symb. Logic 33*, 471–472.

Newman, M. H. A. 1942. On theories with a combinatorial definition of "equivalence." *Annals of Math. 43*, 2nd series, 223–243.

Petznick, G. W. 1970. "Combinatory Programming." Ph.D. thesis, Madison, Univ. of Wisconsin. University Microfilms Publication 70-24812.

Plotkin, G. D. 1972. "A Set-Theoretical Definition of Application." Memorandum MIP-R-95, School of Artificial Intelligence, Univ. of Edinburgh, 32 pp.

Post, E. L. 1936. Finite combinatory processes. Formulation I. *J. Symb. Logic 1*, 103–105.

Post, E. L. 1943. Formal reductions of the general combinatorial decision problem. *Amer. J. Math. 65*, 197–215.

Rosen, B. K. 1973. Tree manipulation systems and Church-Rosser theorems. *JACM 20*, 160–187.

Rosser, J. B. 1935. A mathematical logic without variables. *Annals of Math. 36*, 2nd series, 127–150; *Duke Math. J. 1*, 328–355.

Rosser, J. B. 1956. Review of Curry (1952), "A new proof of the Church-Rosser theorem." *J. Symb. Logic 21*, 377.

Schönfinkel, M. 1924. Uber die Bausteine der mathematischen Logik. *Math. Ann. 92*, 305–316.

Schroer, D. E. 1965. "The Church-Rosser Theorem." Ph.D. thesis, Cornell Univ. University Microfilms Publication 66-41.

Scott, D. 1970. Outline of a mathematical theory of computation. *Proc. 4th Ann. Princeton Conf. on Information Sciences and Systems*, pp. 169–176. (Also in Engeler 1971.)

Smullyan, R. M. 1961. Theory of formal systems. *Annals of Math. Studies 47*, Princeton Univ. Press.

Steel, J. B., Jr. (ed.). 1966. Formal language description languages for computer programming. *Proc. IFIP Working Conference on Formal Language Description Languages*, Amsterdam, North-Holland.

Titgemeyer, R. 1961. Über einen Widerspruch in Cogan's Darstellung der Mengenlehre, *Zeit. Math. Logik Grundlagen Math. 7*, 161–163.

Turing, A. M. 1936; 1937. On computable numbers, with an application to the *Entscheidungsproblem. Proc. London Math. Soc. 42*, 230–265; A correction, *43*, 544–546.

Turing, A. M. 1937. Computability and λ-definability. *J. Symb. Logic 2*, 153–163.

Turner, D. A. 1979*a*. A new implementation technique for applicative languages. *Software—Practice and Experience 9*, 31–49.

Turner, D. A. 1979*b*. Another algorithm for bracket abstraction. *J. Symb. Logic 44*, 267–270.

van Heijenoort, J. 1967. *From Frege to Gödel.* Cambridge, Harvard Univ. Press.

Whitehead, A. N., and B. Russell. 1925. *Principia Mathematica.* Cambridge, Cambridge Univ. Press.