Curry's anticipation of the types used in programming languages*

Jonathan P. Seldin
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
jonathan.seldin@uleth.ca
http://people.uleth.ca/∼jonathan.seldin

February 26, 2003

Abstract

This paper shows that H. B. Curry anticipated both the kind of data types used in computer programming languages and also the dependent function type.

1 Introduction

Types are used in computer programming for at least two purposes:

1. *The interpretation of stored data.* The data stored in the memory of a computer is a string of 0's and 1's. For each such string, there are a number of ways to interpret it. Consider, for example, the string

   10111110011010000000000000000000.

   This string can be interpeted as:

*This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

(a) An unsigned integer. This is just a binary integer. The value is $2^{19} + 2^{21} + 2^{22} + 2^{25} + 2^{26} + 2^{27} + 2^{28} + 2^{29} + 2^{31} = 3,194,486,784$.

(b) A signed integer. The first bit, 1, is the $-$ sign. The value is $-(2^{19} + 2^{21} + 2^{22} + 2^{25} + 2^{26} + 2^{27} + 2^{28} + 2^{29}) = -1,047,003,136$.

(c) A floating point real. The first bit, 1, is -. The next 8 bits, 01111100, are binary for the exponent, which is $124 - 128 = -4$. The remaining bits are the mantissa, which is

$$\begin{aligned} & 11010000000000000000000 \\ = \;& 0.1101_2 = 2^{-1} + 2^{-2} + 2^{-4} \\ = \;& \frac{13}{16} = 0.8125_{10} \end{aligned}$$

So the value is -0.8125×2^{-4}

Types, such as `nat, int, real, bool` are used to tell a compiler which scheme is to be used to interpret the data.

2. *The prevention of programming errors.* In a computer language with strong typing, certain kinds of programming errors will be caught by the compiler before the program is run. If a value being used by a program is not of the type that the compiler expects, the error is detected and the compilation fails.

In many programming languages, variables must be declared with their types, as in `num:int, radius:real, cond:bool`. We also need compound types, for example the condition `radius` > 1, which is a function from real numbers to the Boolean type, has type `real` $\rightarrow$ `bool`.

The kind of types used here are different from the logical types used by Russell and Whitehead [30][1], in which the types are natural numbers. There has been a change in the notion of type from the logical types of Russell and Whitehead to the types now in use in computer science. In this paper, I will discuss the way this newer notion of type began in the late 1920s with H. B. Curry.[2]

In order to fully understand Curry's contribution, it is necessary to understand the formalism in which he worked: combinatory logic. Combinatory

[1]The types were actually introduced by Russell tentatively in [25, Appendix B] and more fully in [26].

[2]But none of this was published until the 1930s.

logic is easier to understand if one first understands its variant, λ-calculus. The next two sections of this paper are background: the first of them will discuss λ-calculus and the second combinatory logic. The last section will discuss the history of Curry's work on the subject.

I would like to thank Roger Hindley and Martin Bunder for their helpful comments and suggestions.

2 Background: λ-calculus

When we first teach students about functions and the function notation, we explain that if $f(x) = x^2$, then to apply this function to a number such as 3, we substitute 3 for x and follow the instructions: $f(3) = 3^2 = 9$. It has recently become common to refer to this function f as $x \mapsto x^2$.

So why do we not write

$$(x \mapsto x^2)(3) = 3^2 = 9?$$

We do not write this because in the 1930s and 1940s, Alonzo Church [4][3] had already introduced the notation

$$(\lambda x \,.\, x^2)3 = 3^2 = 9.$$

This will do for functions of one argument, but what about functions of more than one argument? If we allow the value of a function to be a function, functions of one variable are sufficient. The procedure for this is known as *currying*, after H. B. Curry, who was one of the people to use it extensively.[4] To see how it works, consider the function $f(x, y) = x - y$ of two arguments. We can obtain this function from functions of one argument as follows: let $\mathsf{curry}f$ be a function which, when applied to an argument x, returns the function which subtracts its argument from x. If x is taken to be 3, this would give us

$$(\mathsf{curry}f)3 = \lambda y \,.\, 3 - y.$$

Applying $\mathsf{curry}f$ to x, and then applying the resulting function to a second argument y returns the value of the original function of two arguments:

$$((\mathsf{curry}f)x)y = f(x, y).$$

[3]Actually, Church's original notation in [1, 2] was slightly different. However, the notation of [4] soon became standard in λ-calculus.

[4]But Curry was not the first to use the procedure, and he objected to the use of his name in [16]. However, the use of his name now seems too well established to be changed.

In λ-calculus and combinatory logic, all functions are assumed to take one argument.

On the other hand, it is also assumed that everything is a function. We get the following formal λ-calculus:[5]

Definition 1 Assume we are given an infinite sequence of *variables* $x, y, z, u, v, w, x_1, \ldots$ and perhaps some *constants*. Then λ-*terms* are defined by induction as follows:

1. Every constant and every variable is a term. Constants and variables are called *atoms*.

2. If M and N are terms, then (MN) is a term, called the *application* of M to N. (Parentheses are omitted by association to the left, so that $MNPQ$ stands for $(((MN)P)Q)$.)

3. If v is a variable and M is a term, then $\lambda v \,.\, M$ is a term, called the *abstraction* of M with respect to v. (Parentheses will be omitted so that $\lambda x_1 x_2 \ldots x_n \,.\, M$ stands for $(\lambda x_1 \,.\, (\lambda x_2 \,.\, (\ldots (\lambda x_n \,.\, M) \ldots)))$.)

An occurrence of a variable v inside a subterm of the form $\lambda v \,.\, M$ is said to be *bound*; all other occurrences are said to be *free*. The *substitution* of a term N for a variable v in a term M, $[N/x]M$ is defined in the usual way except that if a variable occurring free in N would become bound in M after the replacement, that bound variable in M is changed to a variable not used elsewhere in the term before the replacement.

Contractions are defined as replacements of subterms of the following kinds:

(α) $\lambda x \,.\, M$ by $\lambda y \,.\, [y/x]M$.

(β) $(\lambda x \,.\, M)N$ by $[N/x]M$.

A (β-)*reduction* is a sequence of zero or more contractions. That M reduces to N is denoted

$$M \rhd N.$$

A (β-)*conversion* is a sequence of zero or more contractions or reverse contractions. That M converts to N is denoted

$$M =_* N.$$

[5]For more details on the pure λ-calculus, see [19, Chapter 1].

Since every term can be both a function and an argument, the λ-calculus cannot be interpreted in the usual way functions are interpreted in set theory. It is probably better to think of λ-terms as representing states of a computer as it is carrying out a program. There are terms which represent infinite loops, for example

$$(\lambda x \,.\, xx)(\lambda x \,.\, xx),$$

which reduces only to itself, and

$$(\lambda x \,.\, xxx)(\lambda x \,.\, xxx),$$

which reduces to

$$(\lambda x \,.\, xxx)(\lambda x \,.\, xxx)(\lambda x \,.\, xxx),$$

etc., and thus represents an infinite expanding loop. In terms of the usual set-theoretic interpretation of functions, these terms are meaningless.[6]

These meaningless terms can be avoided, and λ-terms can be brought into line with the usual set-theoretic notion of function, by assigning *types*.[7] The types involved are not the numerical types of Bertrand Russell, but are rather the kind of types used in programming languages.

Definition 2 Assume that we are given some *atomic types*, $\theta_1, \theta_2, \ldots$. *Types* are defined as follows:

1. Every atomic type is a type.[8]

2. If α and β are types, then $\alpha \to \beta$ is a type.

Types can then be assigned to λ-terms as follows:[9]

Definition 3 The system of *type assignment to λ-terms* assigns types to certain terms by means of rules. That a term M is assigned a type α is indicated by $M : \alpha$. Assumptions assign types to variables. Types are assigned

[6] But there are set-theoretic models of λ-calculus. See [19, Chapters 10–12] for details and further references.

[7] This use of types in connection with λ-terms began with [3].

[8] In [3], the atomic types are o for propositions and ι for individuals. Furthermore, Church's notation for $\alpha \to \beta$ is $(\beta\alpha)$.

[9] Church did not assign types to terms that were already formed, but required that the types match properly when terms were formed. For details see [19, Chapter 13]. Type assignment in the style of Definition 3 is really due to Curry, as we shall see below. For more details on type assignment to λ-terms, see [19, Chapter 15].

to compound terms (terms that are not atoms) by means of the following natural deduction rules (in the style of [21]):

$$(\to \text{e}) \qquad \frac{M : \alpha \to \beta \quad N : \alpha}{MN : \beta}$$

$$(\to \text{i}) \qquad \frac{\begin{array}{c}[x : \alpha] \\ M : \beta\end{array}}{\lambda x \,.\, M : \alpha \to \beta}$$

Condition: x does not occur free in α or in any undischarged assumption.

3 Background: Combinatory Logic

Since Curry worked in combinatory logic rather than λ-calculus, we need to take a look at combinatory logic.[10]

In λ-calculus, there are two operations for forming non-atomic terms: application and abstraction, and the operation of application involves all the complications of bound variables. In combinatory logic, the only primitive operation for forming compound terms is application. Instead of taking abstraction as a primitive term-forming operation, there are special constants in terms of which an abstraction operator can be defined metatheoretically.

Definition 4 Assume we are given an infinite sequence of *variables*, $x, y, z, u, v, w, x_1, \ldots$ and *constants* $\mathsf{I}, \mathsf{K}, \mathsf{S}$ and perhaps others. *Terms* are defined as follows:

1. Every variable and constant is a term. Variables and constants are called *atoms*.
2. If M and N are terms, then (MN) is a term. (Parentheses are omitted as in clause 2 of Definition 1.)

A *contraction* is a replacement of

$$\begin{array}{rcl} \mathsf{I}X & \text{by} & X, \\ \mathsf{K}XY & \text{by} & X, \\ \mathsf{S}XYZ & \text{by} & (XZ)(YZ). \end{array}$$

[10] Combinatory logic began with Schönfinkel [27] and continued with Curry [5, 6, 7, 8, 9, 11].

A *(weak) reduction* is a sequence of zero or more contractions. That M reduces to N is denoted

$$M \rhd N.$$

A *(weak) conversion* is a sequence of zero or more contractions and reverse contractions. That M converts to N is denoted

$$M =_* N.$$

In combinatory logic, all variables are free, so substitution is simple replacement. The equivalence with λ-calculus is obtained by defining abstraction:

Definition 5 The *abstraction* of a term M with respect to a variable x, $[x]M$, is defined by induction on the structure of M as follows, where c stands for any atom different from x and $M \equiv N$ means that M and N are the same term:

$$\begin{aligned} [x]x &\equiv \mathsf{I}, \\ [x]c &\equiv \mathsf{K}c, \\ [x](MN) &\equiv \mathsf{S}([x]M)([x]N). \end{aligned}$$

Note that $[x]M$ is an abbreviation for a term in which x does not occur at all. Hence, it is trivial that $[x]M \equiv [y][y/x]M$. The following can be shown by induction on the structure of M:

Theorem 1 *If M and N are any terms and x any variable,*

$$([x]M)N \rhd [N/x]M.$$

This shows the close relationship between combinatory logic and λ-calculus.[11]

The constants $\mathsf{I}, \mathsf{K}, \mathsf{S}$ are called *basic combinators*. Other combinators can be defined in terms of them. Some examples of such combinators are

$$\begin{aligned} \mathsf{B}XYZ &\rhd X(YZ), \\ \mathsf{C}XYZ &\rhd XZY, \\ \mathsf{W}XY &\rhd XYY. \end{aligned}$$

Types can be assigned to terms in combinatory logic as well as in λ-calculus. The types are the same as for λ-calculus; i.e., the types are defined by Definition 2.[12]

[11] For more details, see [19, Chapters 2 and 7–9].

[12] For more details, see [19, Chapter 14].

Definition 6 The system of *type assignment to combinatory terms* is similar to the system of Definition 3. The difference is that rule $(\to \mathrm{i})$ is replaced by the following axiom schemes:

$$(\to \mathsf{I}) \qquad \mathsf{I} : \alpha \to \alpha$$

$$(\to \mathsf{K}) \qquad \mathsf{K} : \alpha \to (\beta \to \alpha)$$

$$(\to \mathsf{S}) \qquad \mathsf{S} : (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$$

It is possible to derive the equivalent rule to $(\to \mathrm{i})$ as a derived rule:

Theorem 2 *If x does not occur free in A or in any undischarged assumption, then the following holds as a derived rule:*

$$\frac{\begin{array}{c}[x:\alpha]\\ M:\beta\end{array}}{[x]M:\alpha\to\beta}$$

The proof is similar to the proof of the deduction theorem in a system of propositional calculus in which the only primitive rule is modus ponens.

Note that the types can be interpreted as well-formed formulas whose only connective is implication, and the terms can be interpreted as deductions in the fragment of intuitionistic propositional logic in which these are the only formulas. This correspondence is known as the *propositions-as-types* correspondence, or as the *Curry-Howard isomorphism*.[13]

4 Curry's Introduction of Types

In his early work on combinatory logic [5, 6, 7, 8, 9, 11], Curry saw himself building a system of mathematical logic. In this respect, his early work was like that of Church in [1, 2]. It was in this context that he first had the idea for what became his theory of *functionality*. What we now call the type was to be a *predicate*, so that what we now write $M:\alpha$ was, for Curry, αM, interpreted as the predicate α with M as its argument. What we now write as

$$M : \alpha \to \beta,$$

[13] But others besides Curry and Howard came up with this idea independently: they include H. Läuchli, N. G. de Bruijn, and Dana Scott. See [19, §14D, p. 194] for references.

which says that M is a function from α to β, Curry expressed as

$$(\forall x)(\alpha x \supset \beta(Mx)).$$

Statements of this form were to be proved from the axioms and rules of logic.

Curry based much of his logic on a constant Ξ with the property that ΞXY meant essentially $(\forall x)(Xx \supset Yx)$.[14] For this, Curry defined

$$\begin{aligned} \mathsf{F} &\equiv \lambda xyz\,.\,(\forall u)(xu \supset y(zu)) \\ &=_* \lambda xyz\,.\,(\forall u)(xu \supset \mathsf{B}yzu) \\ &=_* \lambda xyz\,.\,\Xi x(\mathsf{B}yz). \end{aligned}$$

Then $\mathsf{F}\alpha\beta f$ stood for what we now write as $f:\alpha \to \beta$.

Curry first introduced this idea in notes dated December 13, 1928, which are reproduced in Appendix A. Note that on the first page, Curry's original notation for $\mathsf{F}\alpha\beta f$ was $F_{\alpha \to \beta} f$. Note that on the second page, Curry first proposed to change this to $[\alpha \to \beta]f$, which is closer to our modern notation, but decided instead to settle for $F\alpha\beta f$.[15]

With these definitions, Curry's version of Rule ($\to$ e), which he called "Rule F," was

$$\frac{\mathsf{F}XYZ \quad XU}{Y(ZU).}$$

He used this rule with axioms for the "types"[16] of the basic combinators:[17]

(FI) $$\mathsf{F}\alpha\alpha\mathsf{I},$$

(FK) $$\mathsf{F}\alpha(\mathsf{F}\beta\alpha)\mathsf{K},$$

(FS) $$\mathsf{F}(\mathsf{F}\alpha(\mathsf{F}\beta\gamma))(\mathsf{F}(\mathsf{F}\alpha\beta)(\mathsf{F}\alpha\gamma))\mathsf{S}.$$

This, together with his rule for conversion

[14]The main difference in meaning between ΞXY and $(\forall x)(Xx \supset Yx)$ is a technical matter of when each has logical significance as a proposition.

[15]The 'F' became 'F' only in [17].

[16]He called them *functional characters*."

[17]The rules are stated here for the basic combinators I, K, and S. However, Curry originally did not take these as his basic combinators; instead he took B, C, K, and W. This is mainly because he did not understand the role of S in the definition of abstraction until May, 1942, when he first gave in his notes the definition of abstraction of Definition 5 above. Before that, he used a completely different definition of abstraction. The definition of abstraction given in Definition 5 first appeared in print in 1949 in [14].

$$\text{(Eq)} \qquad \frac{X \quad X =_* Y}{Y.}$$

is Curry's *theory of functionality*.

Early on, Curry noticed a similarity between the types of combinators and implication formulas of propositional calculus, and he soon started using names for these implication formulas which came from this similarity:

$$\text{(PI)} \qquad A \supset A,$$

$$\text{(PK)} \qquad A \supset (B \supset A),$$

$$\text{(PS)} \qquad (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)),$$

Curry was giving names somewhat like this as early as July 15, 1930; see his notes of this date reproduced in Appendix B. The original names were "P_k instead of '(PK)', but he was using names of the latter form within a couple of years.[18] This is probably the origin of the propositions-as-types notion.[19]

Curry wrote his first paper on the theory of functionality in 1932, but he had trouble getting it accepted for publication.[20] While he was trying to get the full paper accepted, he wrote an extended abstract which appeared in 1934 as [10]. The full paper finally appeared in 1936 as [12].

Meanwhile, in 1935, Kleene and Rosser [20] proved inconsistent the original systems of both Church and Curry. Church reacted by abstracting the pure λ-calculus from his system, and he and his students (Kleene and Rosser)

[18]The faint left-hand column, using the more modern version of the names, is written in pencil, and may date from July 19, 1936, which is when he classified the notes with a card in his card index.

[19]But Curry never carried this idea beyond a kind of curiosity; i.e., a means to find proofs of pure implication formulas in the intuitionistic propositional logic. It was the others to came up with this idea indepently who gave the idea its full development.

[20]One of the referees who originally rejected the paper was Alonzo Church. When I was Curry's research assistant (as a graduate student), I saw in his files a typed referee's report on this paper with a handwritten correction that was rather clearly in Church's handwriting. Curry once told me that he was disappointed when Church later published [3], because he recognized that Church's types were closely related to his functional characters. In fairness to Church, Curry's early theory of functionality did not really look at all like Church's type theory, and, in fact, only came to look like it in the 1950s, with the *basic* theory of functionality. See below.

lost interested in building a system of logic using it as a basis. Curry's reaction was different: he had already considered the possibility that his original system might be inconsistent, and he proposed to develop one or more consistent systems by using the theory of functionality with a constant H (standing for "proposition") to restrict the terms that could occur as arguments of logical operators. He also developed, in the late 1930s, a proposal for a sequence of kinds of systems that differed according to which of the operators F, Ξ, P (implication), and Π (universal quantifier) is to be primitive. There were three such kinds:

1. $\mathcal{F}_1$: primitive is F. $\Xi \equiv \lambda xy \,.\, \mathsf{F}xy\mathsf{I}$ or $\Xi \equiv \lambda xy \,.\, \mathsf{F}x\mathsf{I}y$.
2. $\mathcal{F}_2$: primitive is Ξ. $\mathsf{F} \equiv \lambda xyz \,.\, \Xi x(\mathsf{B}yz)$, $\mathsf{P} \equiv \lambda xy \,.\, \Xi(\mathsf{K}x)(\mathsf{K}y)$, and $\Pi \equiv \lambda x \,.\, \Xi\mathsf{E}x$, where $\vdash \mathsf{E}X$ for all terms X.
3. $\mathcal{F}_3$: primitives are P and Π. $\Xi \equiv \lambda xy \,.\, \Pi(\lambda u \,.\, \mathsf{P}(xu)(yu))$.

Curry originally thought that on reasonable assumptions, $\mathcal{F}_1$ systems are weaker than $\mathcal{F}_2$ systems which, in turn, are weaker than $\mathcal{F}_3$ systems; see [13]. However, it has turned out that all three kinds of systems are of essentially equal strength.

In the early 1950s, Curry used his idea that systems of the kind $\mathcal{F}_1$ are weaker than the others by trying hard to prove that a system of this kind is consistent if there are no restrictions on the terms that can be types. By 1955, he discovered that this is not true; see [15] and [17, §**10**A3]. The proof of inconsistency shows a particular problem with rule (Eq) in the theory of functionality: Curry derives

$$\vdash \beta(\mathsf{WWW}),$$

where β is an arbitrary term. He then substitutes $\mathsf{K}X$ for β, where X is an arbitrary term, in order to conclude

$$\vdash X.$$

Inferences like this show that the theory of functionality at this stage was still not exactly like a system of types.

Curry's first response to this discovery was to define the *basic* theory of functionality. In this theory, the only terms which can be types are those defined from the atomic types by the operation of forming $\mathsf{F}\alpha\beta$ from α and

β. All of these types are terms in normal form, so all inferences by rule (Eq) take the form

$$\frac{\alpha M \quad M =_* N}{\alpha N.}$$

Not surprisingly, this systems is consistent; see [17, Chapter 9]. Curry also defined *F-deductions*, in which conversion rules are not used. Later, in about 1966, Curry split Rule (Eq) into two separate rules, one for terms and the other for types:

$$\text{(Eqs)} \quad \frac{\alpha M \quad M =_* N}{\alpha N,} \qquad\qquad \text{(Eqp)} \quad \frac{\alpha M \quad \alpha =_* \beta}{\beta M.}$$

These separated rules finally convert the theory of functionality into modern type assignment. See [18, Chapter **14**].[21]

In the 1950s, Curry had long been thinking about possible generalizations of the theory of functionality. On March 29, 1956, he had an idea which is reproduced in Appendix C. With this idea, we have

$$\begin{aligned} \mathsf{G} \quad &\equiv \quad \lambda xyz \,.\, \Xi x(\mathsf{S}yz) \\ &=_* \quad \lambda xyz \,.\, (\forall u)(xu \supset \mathsf{S}yzu) \\ &=_* \quad \lambda xyz \,.\, (\forall u)(xu \supset yu(zu)). \end{aligned}$$

Curry's Rule G is the elimination rule, which is

$$\frac{\mathsf{G}ABM \quad AN}{BN(MN).}$$

The introduction rule is

$$\frac{\begin{array}{c}[Ax] \\ BM\end{array}}{\mathsf{G}A(\lambda x \,.\, B)(\lambda x \,.\, M),}$$

where x does not occur free in A or in any undischarged assumption. The conversion rules are the separated rules, and in some systems only (Eqp) is postulated. With these separated conversion rules, it is possible to re-write

[21] This chapter was written by revising [28, Chapter 3] from my dissertation, but I took the idea for the separated conversion rules from Curry's draft for [18].

the rules as follows:

(Ge) $$\frac{M:\mathsf{G}ABN:A}{MN:BN,}$$

(Gi) $$\frac{\begin{array}{c}[x:A]\\M:B\end{array}}{(\lambda\,.\,M):(\lambda x\,.\,B),}$$ *Condition:* x does not occur free in A or in any undischarged assumption.

(Eqp) $$\frac{M:A\quad A=_*B}{M:B,}$$

(Eqs) $$\frac{M:A\quad M=_*N}{N:A.}$$

Rule (Eqs) may be omitted.[22]

This form of the rules makes it clear that Curry's $\mathsf{G}AB$ is the *dependent function type*, which is nowadays usually written $(\Pi x:A)(Bx)$ or $(\forall x:A)(Bx)$ or $(\Pi x:A\,.\,Bx)$. Thus, Curry anticipated the dependent function type as well as the types of programming languages.

References

[1] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33:346–366, 1932.

[2] Alonzo Church. A set of postulates for the foundation of logic (second paper). *Annals of Mathematics*, 34:839–864, 1933.

[3] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[4] Alonzo Church. *The Calculi of Lambda Conversion*. Princeton University Press, 1941.

[5] Haskell Brooks Curry. An analysis of logical substitution. *American Journal of Mathematics*, 51:363–384, 1929.

[22] For more details, see [29]. This paper is an abridged version (with most proofs omitted) of a longer manuscript written in the style of a chapter of [18].

[6] Haskell Brooks Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930. Inauguraldissertation.

[7] Haskell Brooks Curry. The universal quantifier in combinatory logic. *Annals of Mathematics*, 32:154–180, 1931.

[8] Haskell Brooks Curry. Some additions to the theory of combinators. *American Journal of Mathematics*, 54:551–558, 1932.

[9] Haskell Brooks Curry. Apparent variables from the standpoint of combinatory logic. *Annals of Mathematics*, 34:381–404, 1933.

[10] Haskell Brooks Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20:584–590, 1934.

[11] Haskell Brooks Curry. Some properties of equality and implication in combinatory logic. *Annals of Mathematics*, 35:849–850, 1934.

[12] Haskell Brooks Curry. First properties of functionality in combinatory logic. *Tôhoku Mathematical Journal*, 41:371–401, 1936.

[13] Haskell Brooks Curry. Some advances in the combinatory theory of quantification. *Proceedings of the National Academy of Sciences of the United States of America*, 28:564–569, 1942.

[14] Haskell Brooks Curry. A simplification of the theory of combinators. *Synthese*, 7:391–399, 1949.

[15] Haskell Brooks Curry. The inconsistency of the full theory of combinatory functionality (abstract). *Journal of Symbolic Logic*, 20:91, 1955.

[16] Haskell Brooks Curry. Some philosophical aspects of combinatory logic. In John Barwise, H. Jerome Keisler, and Kenneth Kunen, editors, *The Kleene Symposium: A Collection of Papers on Recursion Theory and Intuitionism*, pages 85–101. North-Holland, Amsterdam, 1980.

[17] Haskell Brooks Curry and Robert Feys. *Combinatory Logic*, volume 1. North-Holland Publishing Company, Amsterdam, 1958. Reprinted 1968 and 1974.

[18] Haskell Brooks Curry, J. Roger Hindley, and Jonathan P. Seldin. *Combinatory Logic*, volume 2. North-Holland Publishing Company, Amsterdam and London, 1972.

[19] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-calculus*. Cambridge University Press, 1986.

[20] S. C. Kleene and J. B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36:630–636, 1935.

[21] Dag Prawitz. *Natural Deduction*. Almqvist & Wiksell, Stockholm, Göteborg, and Uppsala, 1965.

[22] J. B. Rosser. A mathematical logic without variables, part I. *Annals of Mathematics*, 36:127–150, 1935.

[23] J. B. Rosser. A mathematical logic without variables, part II. *Duke Mathematical Journal*, 1:328–355, 1935.

[24] J. B. Rosser. New sets of postulates for combinatory logics. *Journal of Symbolic Logic*, 7:18–27, 1942.

[25] Bertrand Russell. *The Principles of Mathematics*, volume 1. Cambridge University Press, 1903.

[26] Bertrand Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.

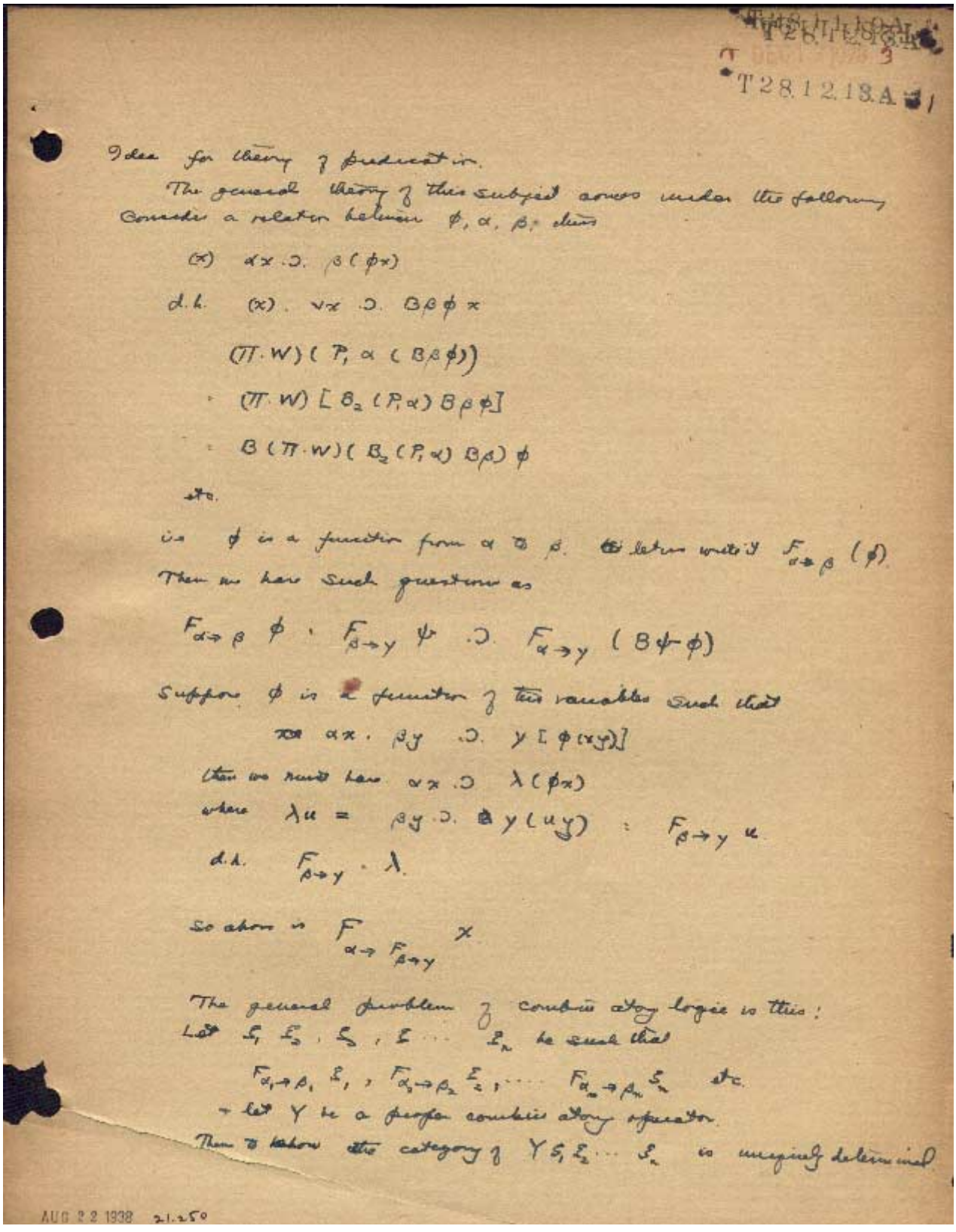
[27] Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.

[28] Jonathan P. Seldin. *Studies in Illative Combinatory Logic*. PhD thesis, University of Amsterdam, 1968.

[29] Jonathan P. Seldin. Progress report on generalized functionality. *Annals of Mathematical Logic*, 17:29–59, 1979.

[30] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, England, 1910–1913. Second edition, 1925–1927.

A Curry's Notes of December 13, 1928



T28.12.18.A

Idea for theory of predicates.

The general theory of this subject comes under the following. Consider a relation between ϕ, α, β: thus

(x) $\alpha x . \supset . \beta(\phi x)$

i.e. (x) . $\forall x . \supset . B\beta\phi x$

$(\Pi \cdot W)(P, \alpha(B\beta\phi))$

$(\Pi \cdot W)[B_2(P,\alpha) B\beta\phi]$

$B(\Pi \cdot W)(B_2(P,\alpha) B\beta)\phi$

etc.

i.e. ϕ is a function from α to β. We letter write it $F_{\alpha\to\beta}(\phi)$.

Then we have such questions as

$F_{\alpha\to\beta}\phi \cdot F_{\beta\to\gamma}\psi . \supset . F_{\alpha\to\gamma}(B\psi\phi)$

Suppose ϕ is a function of two variables such that

$\alpha x \cdot \beta y . \supset . \gamma[\phi(xy)]$

then we must have $\alpha x . \supset \lambda(\phi x)$

where $\lambda u = \beta y . \supset . \gamma(uy) : F_{\beta\to\gamma} u$

i.e. $F_{\beta\to\gamma} \cdot \lambda$.

So above is $F_{\alpha\to F_{\beta\to\gamma}} x$

The general problem of combinatory logic is this:

Let $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma \cdots \Sigma_n$ be such that

$F_{\alpha_1\to\beta_1}\Sigma_1, F_{\alpha_2\to\beta_2}\Sigma_2, \cdots F_{\alpha_n\to\beta_n}\Sigma_n$ etc.

+ let Y be a proper combinatory operator.

Then to know the category of $Y\Sigma_1\Sigma_2\cdots\Sigma_n$ is marginally determined

AUG 2 2 1938 21-250

Instead of $F_{\alpha\to\beta}$ a more convenient notation is

$[\alpha\to\beta]$

and for $F_{\alpha\to F_{\beta\to\gamma}}$ have $[\alpha,\beta\to\gamma]$

etc.

in general if $\phi(x_1\cdots x_n)$ is such that

$\alpha x_1 . \alpha_2 x_2 . \cdots . \alpha_n x_n . \supset . \beta(\phi x_1 x_2 \cdots x_n)$

then $[\alpha_1,\alpha_2,\cdots\alpha_n\to\beta]\ \phi$

then we have $[\alpha,\beta\to\gamma] \equiv [\alpha\to[\beta\to\gamma]]$

and in general $[\alpha_1,\alpha_2,\cdots\alpha_n\to\beta] \equiv [\alpha_1\to[\alpha_2\to[\alpha_3\to\cdots[\alpha_n\to\beta]]]$

postulates are such as

$[\alpha\to\beta]\phi . [\beta\to\gamma]\psi . \supset . [\alpha\to\gamma]\phi(B\psi\phi)$

still holds notation, for $\alpha\to\beta$ use $F\alpha\beta$

then $[\alpha,\beta\to\gamma] = F\alpha(F\beta\gamma)$ &c

we have $F\alpha\beta\phi . F\beta\gamma\psi . \supset . F\alpha\gamma(B\psi\phi)$

definition of F is then

$$\begin{aligned} F\alpha\beta x &= (y)\ \alpha y . \supset . \beta(xy) \\ &= (y)\ \alpha y . \supset . B\beta x y \\ &= (\Pi . W)(P_1\alpha(B\beta x)) \\ &= (\Pi . W)(B B_2 P_1 \alpha B\beta x) \\ &= (\Pi . W)(C_3 B B_2 P_1 B\alpha\beta x) \\ &= B_3(\Pi . W)(C_3 B B_2 P_1 B)\alpha\beta x \end{aligned}$$

$\therefore F = B_3(\Pi . W)(C_3 B B_2 P_1 B)$

From this the various postulates may perhaps be proved
other postulates are such as

$F\alpha(F\beta\gamma)\,x \;.\supset.\; F\beta(F\alpha\gamma)\,(C.x)$

etc. $F\alpha(F\alpha\gamma)\,x \;.\supset.\; F\alpha\gamma\,(Wx)$

The postulate for B may be put in the form.

$F\alpha\beta x \;.\; F\beta\gamma y \;.\supset.\; F\alpha\gamma\,(Bxy)$

i.e. $F(F\alpha\beta)(F(F\alpha\gamma)(F\alpha\gamma))B$

i.e. $F(F\alpha\beta)(F(F\beta\gamma)(F\alpha\gamma))B$

this is not a contradictory proposition about B even if involves the variables α and β.

if $\alpha = \beta^2\gamma$ we have

~~$F(F\alpha\alpha)$~~ $F(F\alpha\alpha)(F(F\alpha\alpha)(F\alpha\alpha))B$

i.e. $F(F\alpha\alpha)(F\alpha\alpha)(WB)$

let α be true proposition.

Then $F\alpha\alpha N$ when N is negation

whence $F\alpha\alpha(WBN)$

but let p be a proposition $WBNp = NNp = p$.

which is no longer

The contradiction arises in $W(BN)$

If we have the postulate for B

$F(F\alpha\beta)(F(F\beta\gamma)(F\alpha\gamma))B$

can we deduce contradict in in $W(BN)$

where $W(BN)x^{\dagger} = BNxx = N(xx)$

we have $F\pi\pi N$. set $\alpha = \beta = \pi$

$F(F\pi\pi)(F(F\pi\gamma)(F\pi\gamma))B$.

we have only $F\pi\pi x \supset F\pi\gamma(BNx)$

d.h. $F(F\pi\gamma)(F\pi\gamma)(BN)$

from which we can conclude nothing).

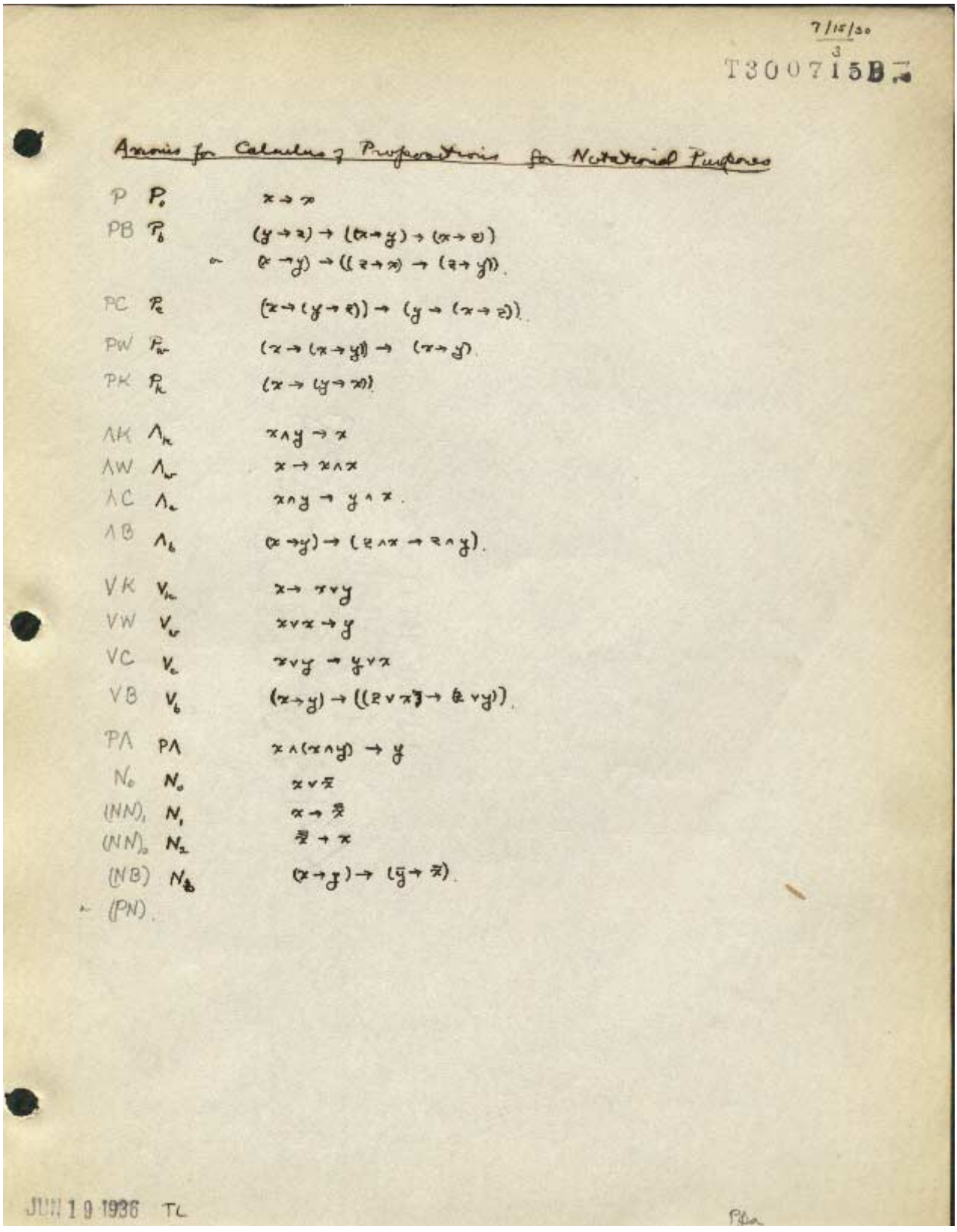
there is γ for $F\beta\alpha(F\beta\gamma)BN$

where $\alpha = F\pi\gamma$

$\beta = \pi$

$\gamma = \gamma$

B Curry's Notes of July 15, 1930



7/15/30

T300715B

Axioms for Calculus of Propositions for Notational Purposes

P	P_0	$x \to x$
PB	P_b	$(y \to z) \to ((x \to y) \to (x \to z))$
		$\vdash (x \to y) \to ((z \to x) \to (z \to y))$.
PC	P_c	$(x \to (y \to z)) \to (y \to (x \to z))$.
PW	P_w	$(x \to (x \to y)) \to (x \to y)$.
PK	P_k	$(x \to (y \to x))$.
ΛK	Λ_k	$x \wedge y \to x$
ΛW	Λ_w	$x \to x \wedge x$
ΛC	Λ_c	$x \wedge y \to y \wedge x$.
ΛB	Λ_b	$(x \to y) \to (z \wedge x \to z \wedge y)$.
VK	V_k	$x \to x \vee y$
VW	V_w	$x \vee x \to y$
VC	V_c	$x \vee y \to y \vee x$
VB	V_b	$(x \to y) \to ((z \vee x) \to (z \vee y))$.
PΛ	PΛ	$x \wedge (x \wedge y) \to y$
N_0	N_0	$x \vee \bar{x}$
(NN)$_1$	N_1	$x \to \bar{\bar{x}}$
(NN)$_2$	N_2	$\bar{\bar{x}} \to x$
(NB)	N_b	$(x \to y) \to (\bar{y} \to \bar{x})$.
$\vdash$ (PN).		

JUN 19 1936 TC

C Curry's Notes of March 29, 1956

T 56.03.29.A

An Idea in Regard to G.

Take formulas as rule,

Rule G: $G\Xi\Sigma\xi, \Xi U \vdash \Sigma U(\xi U)$

Then $G = [x,y,z]\ \Xi x.(\Sigma y z)$.

Then G disjoins from $\mathfrak{B}$ in putting S for B.