Propositions as Types

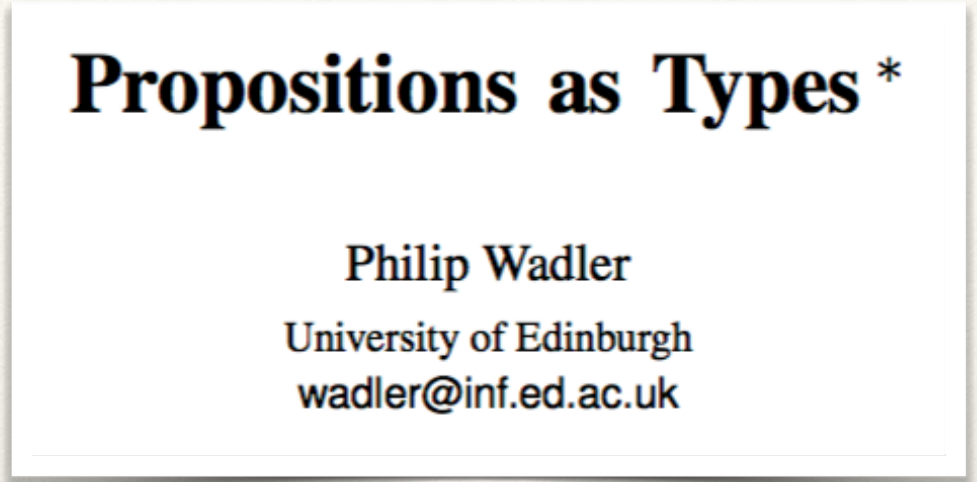
Nikolai Kudasov
Innopolis University

Innopolis University, 2023

An adaptation of Philip Wadler's talk given in St. Louis on August 25th, 2015

Philip Wadler (1956 – now)

Philip Wadler (2015) – Propositions as Types

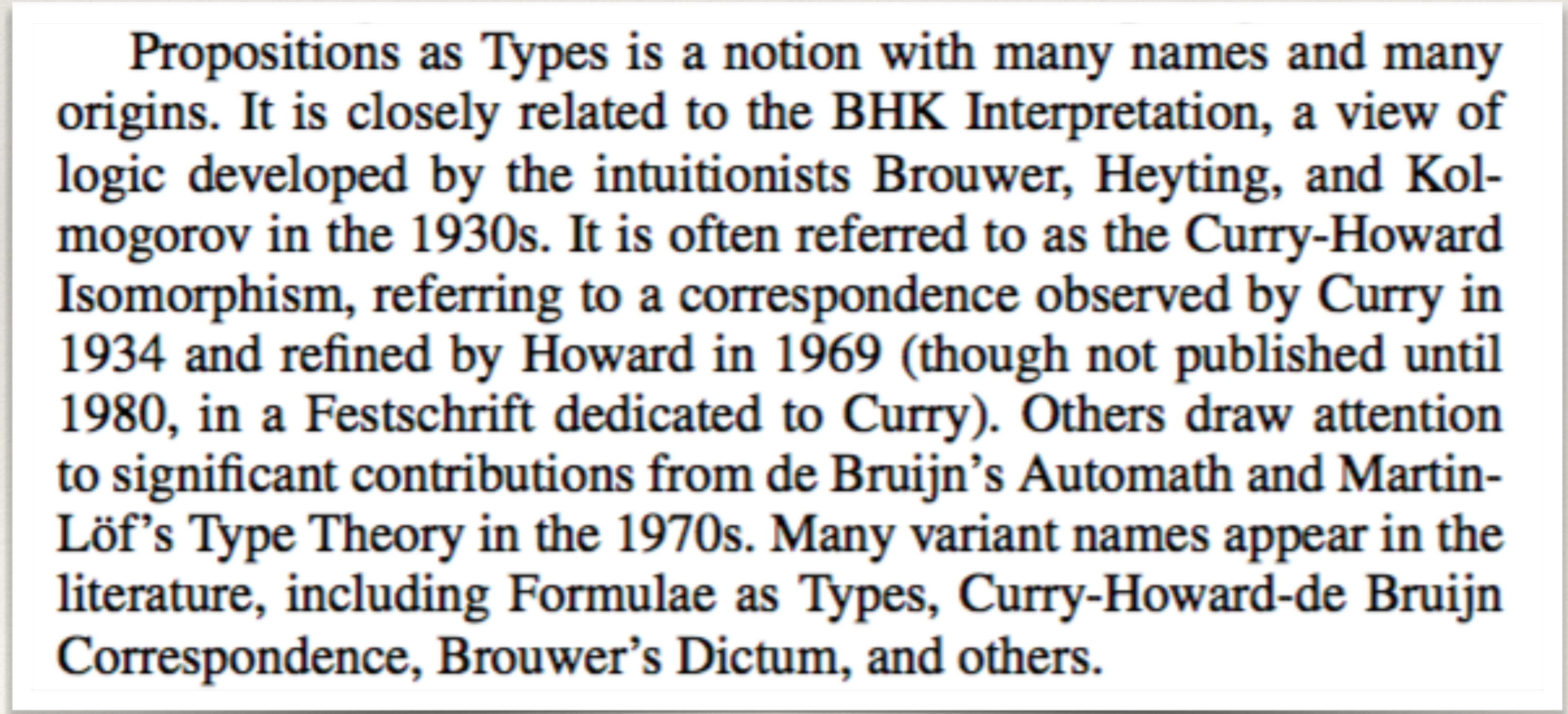


Propositions as Types *

Philip Wadler

University of Edinburgh

wadler@inf.ed.ac.uk



Propositions as Types is a notion with many names and many origins. It is closely related to the BHK Interpretation, a view of logic developed by the intuitionists Brouwer, Heyting, and Kolmogorov in the 1930s. It is often referred to as the Curry-Howard Isomorphism, referring to a correspondence observed by Curry in 1934 and refined by Howard in 1969 (though not published until 1980, in a Festschrift dedicated to Curry). Others draw attention to significant contributions from de Bruijn's Automath and Martin-Löf's Type Theory in the 1970s. Many variant names appear in the literature, including Formulae as Types, Curry-Howard-de Bruijn Correspondence, Brouwer's Dictum, and others.

http://www.cs.bc.edu/~muller/teaching/lc/WadlerPropositionsAsTypes.pdf

Euclid (325–265 BC)

al-Khwārizmī (780–846 AC)

Computability

- **Alonzo Church: Lambda-calculus**
  An unsolvable problem of elementary number theory
  Bulletin the American Mathematical Society, May 1935
- **Kurt Gödel: Recursive functions**
  Stephen Kleene, General recursive functions of natural numbers
  Bulletin the American Mathematical Society, July 1935
- **Alan Turing: Turing machines**
  On computable numbers, with an application to the Entscheidungsproblem
  Proceedings of the London Mathematical Society, received 25 May 1936

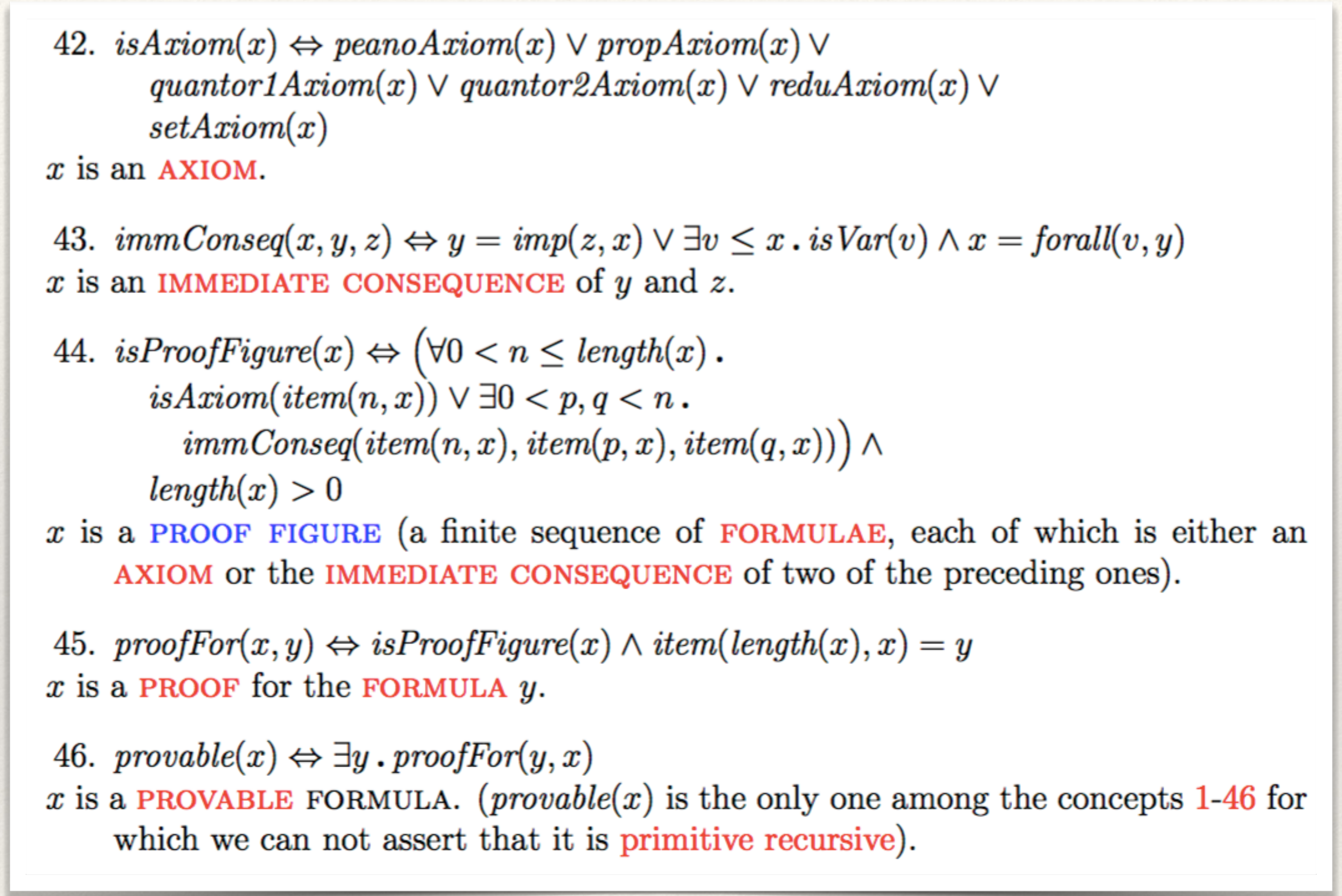
David Hilbert (1862–1943)

David Hilbert (1928) – Entscheidungsproblem

- Find an **algorithm** that would take any decision problem (a **yes or no question** written as a logic proposition *S* in some formal language) and after **finitely many steps** would produce either «**true**» or «**false**», depending on whether *S* is true or false.

Kurt Gödel (1906–1978)

Kurt Gödel (1931) – Incompleteness theorem



42. $isAxiom(x) \Leftrightarrow peanoAxiom(x) \lor propAxiom(x) \lor$
$quantor1Axiom(x) \lor quantor2Axiom(x) \lor reduAxiom(x) \lor$
$setAxiom(x)$

x is an AXIOM.

43. $immConseq(x, y, z) \Leftrightarrow y = imp(z, x) \lor \exists v \leq x \,.\, isVar(v) \land x = forall(v, y)$

x is an IMMEDIATE CONSEQUENCE of y and z.

44. $isProofFigure(x) \Leftrightarrow \big(\forall 0 < n \leq length(x) \,.$
$isAxiom(item(n, x)) \lor \exists 0 < p, q < n \,.$
$immConseq(item(n, x), item(p, x), item(q, x))\big) \land$
$length(x) > 0$

x is a PROOF FIGURE (a finite sequence of FORMULAE, each of which is either an AXIOM or the IMMEDIATE CONSEQUENCE of two of the preceding ones).

45. $proofFor(x, y) \Leftrightarrow isProofFigure(x) \land item(length(x), x) = y$

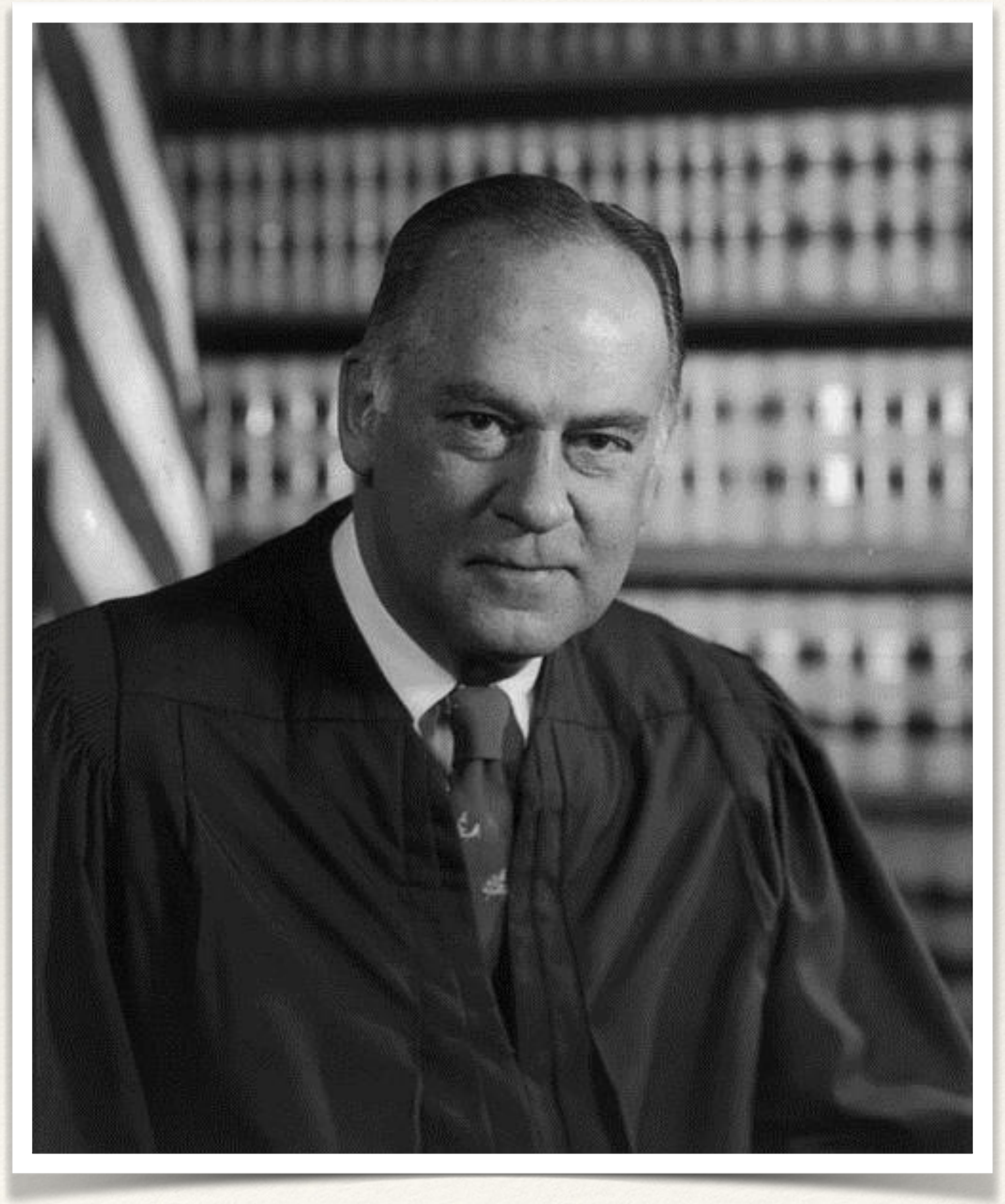
x is a PROOF for the FORMULA y.

46. $provable(x) \Leftrightarrow \exists y \,.\, proofFor(y, x)$

x is a PROVABLE FORMULA. ($provable(x)$ is the only one among the concepts 1-46 for which we can not assert that it is primitive recursive).

«This statement is not provable»

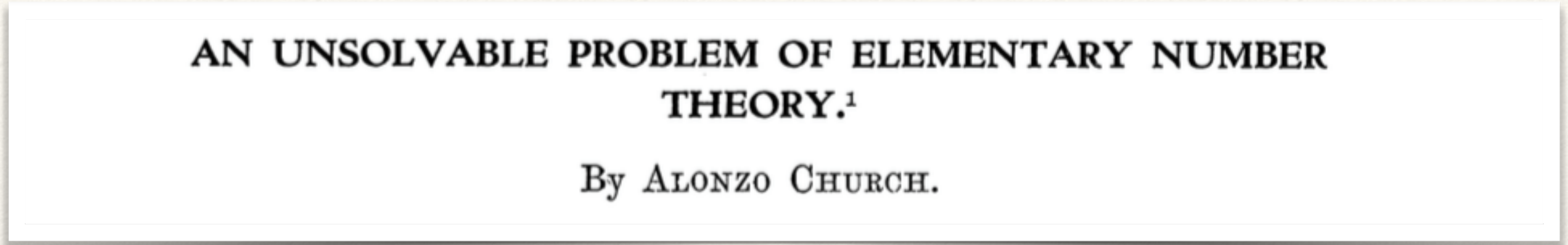
«I Know It When I See It»



18+

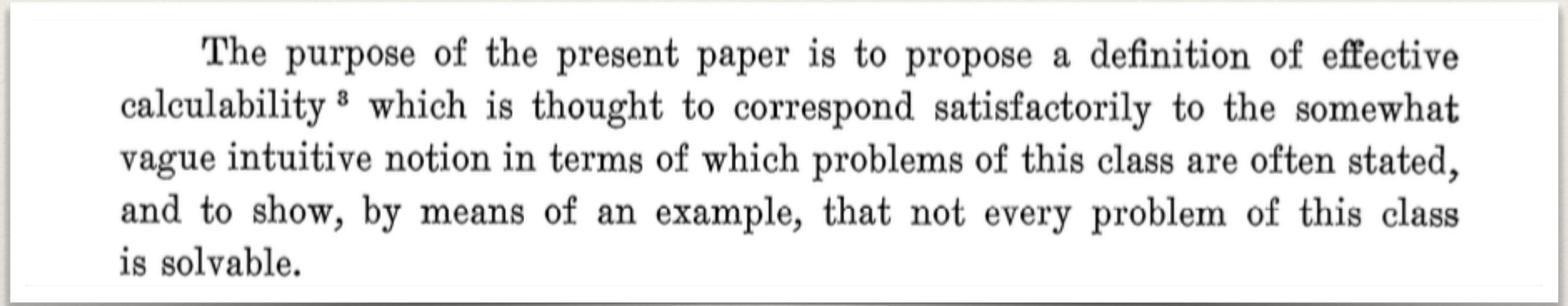
Alonzo Church (1903–1995)

Alonzo Church (1935) – Lambda-calculus

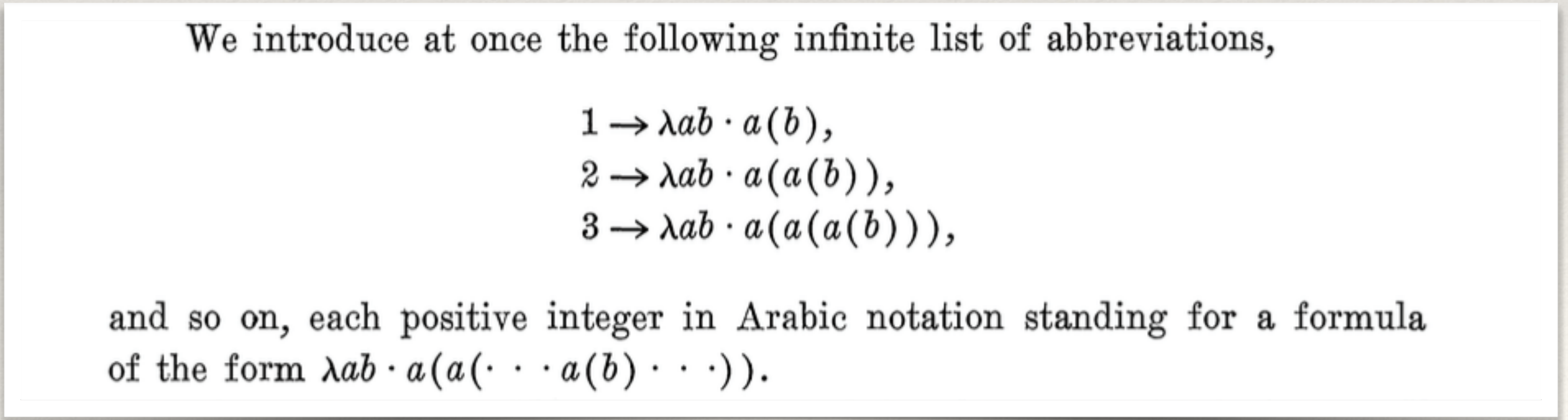


AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.[1]

By Alonzo Church.



The purpose of the present paper is to propose a definition of effective calculability[3] which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which problems of this class are often stated, and to show, by means of an example, that not every problem of this class is solvable.



We introduce at once the following infinite list of abbreviations,

$$1 \to \lambda ab \cdot a(b),$$
$$2 \to \lambda ab \cdot a(a(b)),$$
$$3 \to \lambda ab \cdot a(a(a(b))),$$

and so on, each positive integer in Arabic notation standing for a formula of the form $\lambda ab \cdot a(a(\cdot \cdot \cdot a(b) \cdot \cdot \cdot))$.

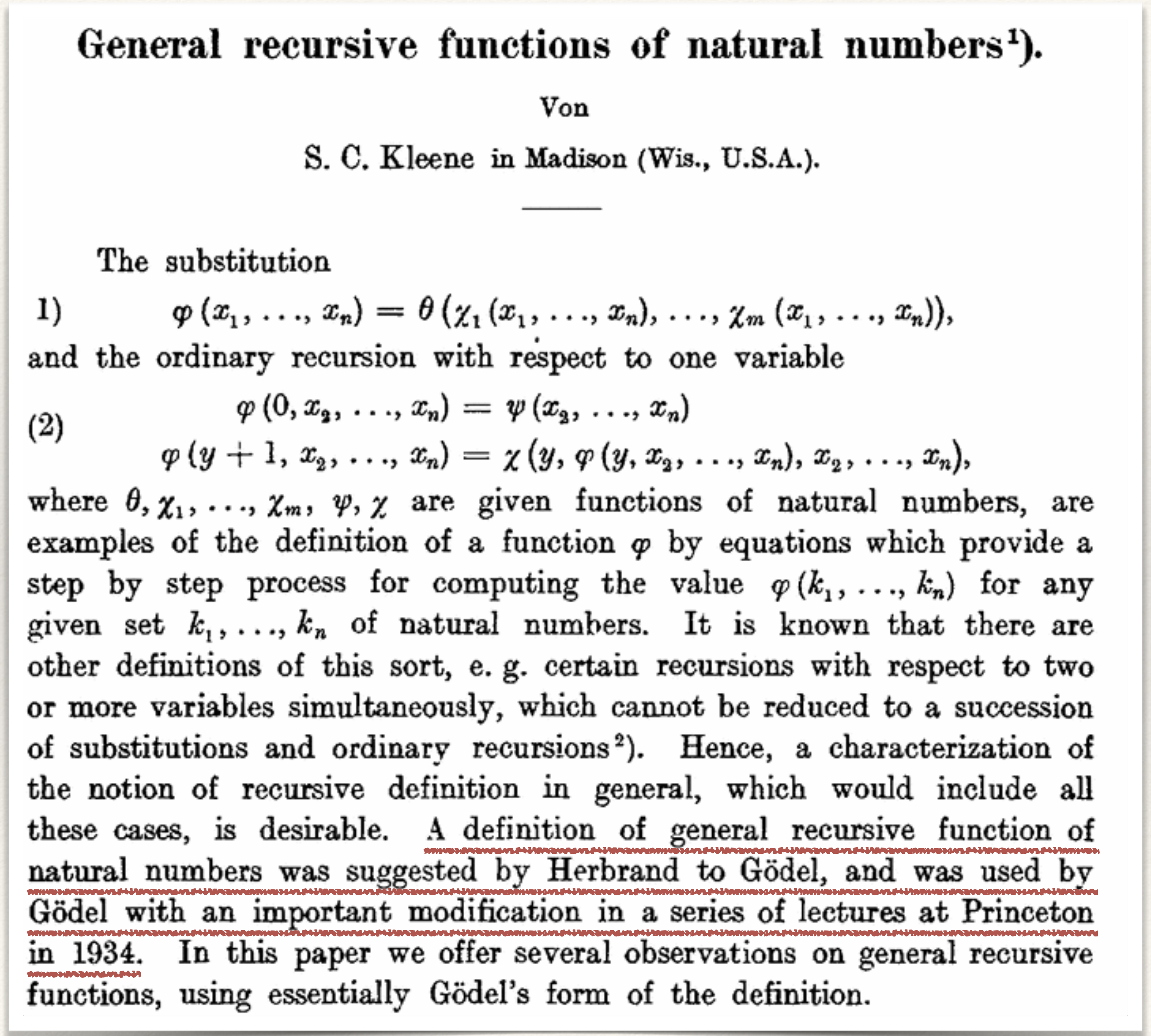
Alonzo Church (1935) – λ-calculus

$$\begin{array}{lcl} L, M, N & ::= & x \\ & | & (\lambda x.\, N) \\ & | & (L\ M) \end{array}$$

Complete syntax for λ-calculus.

Kurt Gödel (1906–1978)

Kurt Gödel (1936) – Recursive functions



General recursive functions of natural numbers[1]).

Von

S. C. Kleene in Madison (Wis., U.S.A.).

The substitution

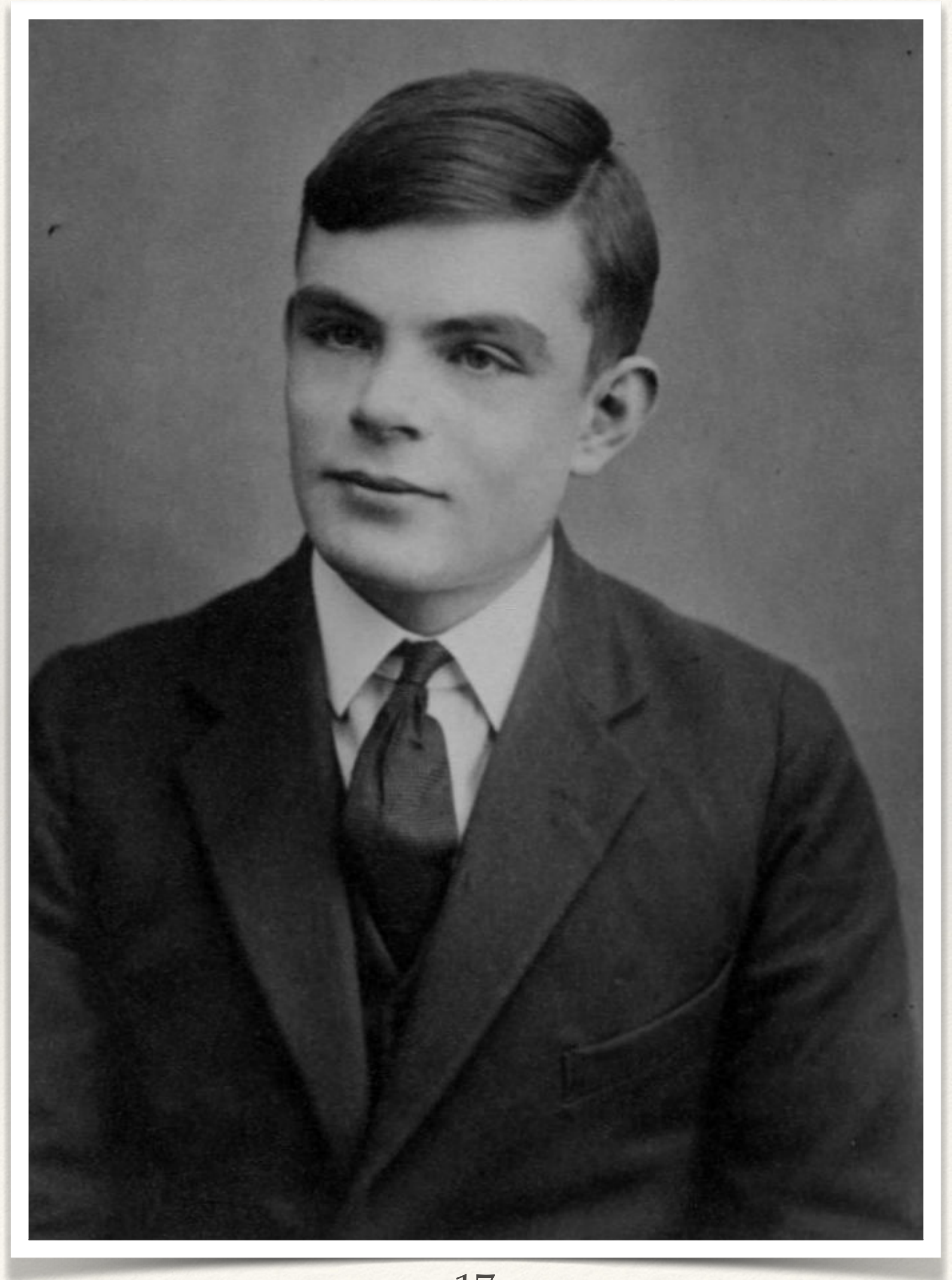
1) $\varphi(x_1, \ldots, x_n) = \theta(\chi_1(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n))$,

and the ordinary recursion with respect to one variable

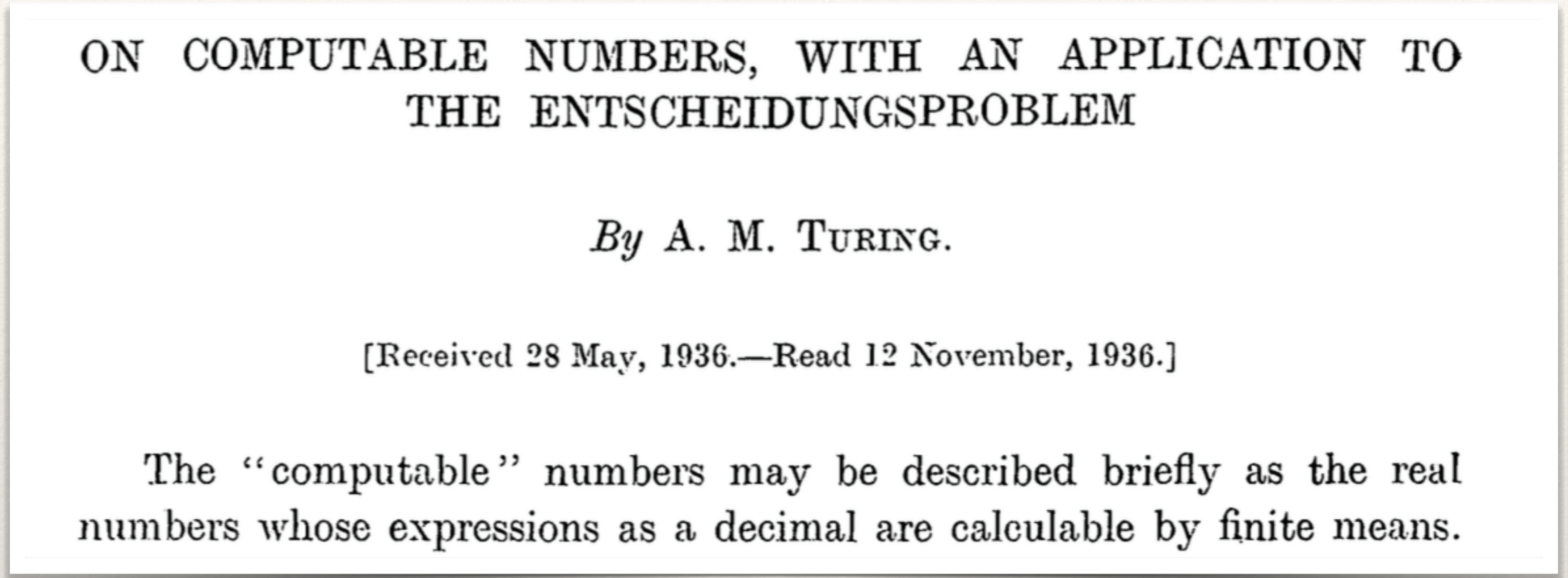
(2)
$$\begin{aligned}\varphi(0, x_2, \ldots, x_n) &= \psi(x_2, \ldots, x_n)\\ \varphi(y+1, x_2, \ldots, x_n) &= \chi(y, \varphi(y, x_2, \ldots, x_n), x_2, \ldots, x_n),\end{aligned}$$

where $\theta, \chi_1, \ldots, \chi_m, \psi, \chi$ are given functions of natural numbers, are examples of the definition of a function φ by equations which provide a step by step process for computing the value $\varphi(k_1, \ldots, k_n)$ for any given set $k_1, \ldots, k_n$ of natural numbers. It is known that there are other definitions of this sort, e. g. certain recursions with respect to two or more variables simultaneously, which cannot be reduced to a succession of substitutions and ordinary recursions[2]). Hence, a characterization of the notion of recursive definition in general, which would include all these cases, is desirable. A definition of general recursive function of natural numbers was suggested by Herbrand to Gödel, and was used by Gödel with an important modification in a series of lectures at Princeton in 1934. In this paper we offer several observations on general recursive functions, using essentially Gödel's form of the definition.

Alan Turing (1912–1954)



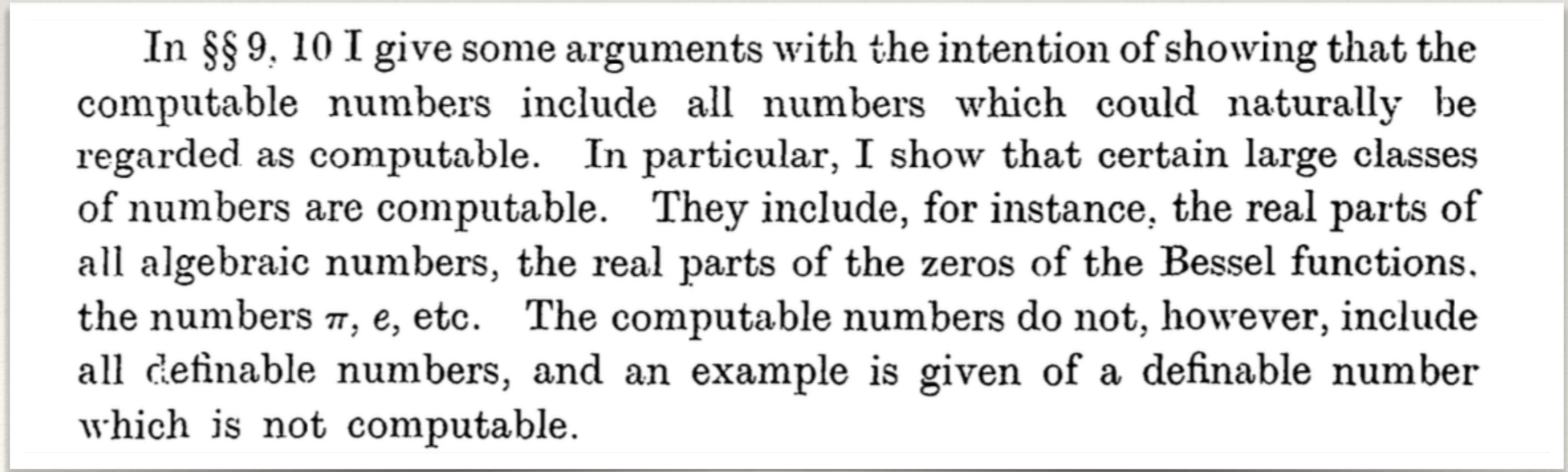
Alan Turing (1936) – Turing machines


ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means.

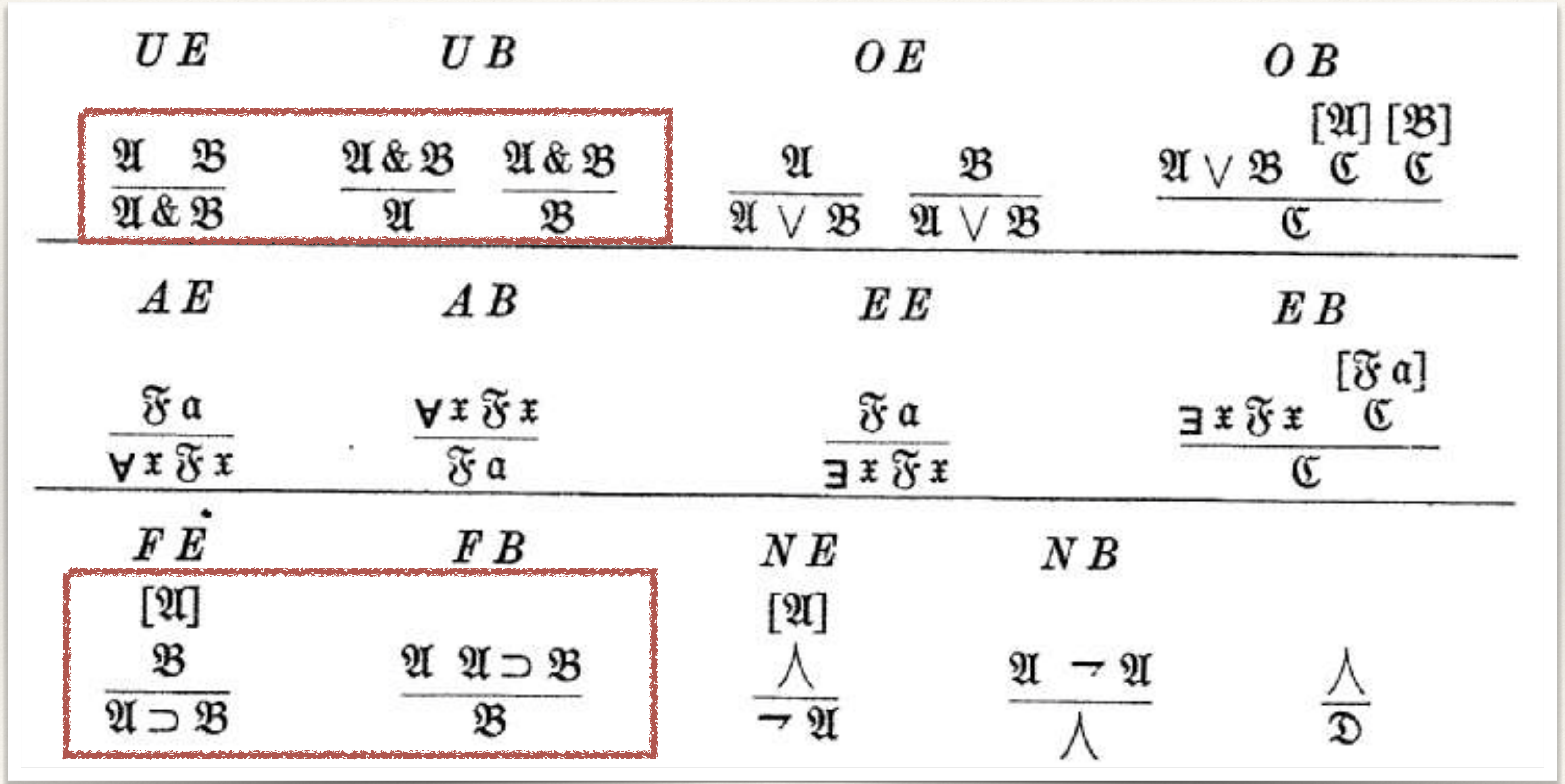

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Part 2

Propositions as Types

Gerhard Gentzen (1909–1945)

Gerhard Gentzen (1935) – Natural deduction



UE	*UB*	*OE*	*OB*
$\dfrac{\mathfrak{A} \quad \mathfrak{B}}{\mathfrak{A} \,\&\, \mathfrak{B}}$	$\dfrac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{A}} \qquad \dfrac{\mathfrak{A} \,\&\, \mathfrak{B}}{\mathfrak{B}}$	$\dfrac{\mathfrak{A}}{\mathfrak{A} \lor \mathfrak{B}} \qquad \dfrac{\mathfrak{B}}{\mathfrak{A} \lor \mathfrak{B}}$	$\dfrac{\mathfrak{A} \lor \mathfrak{B} \quad \overset{[\mathfrak{A}]}{\mathfrak{C}} \quad \overset{[\mathfrak{B}]}{\mathfrak{C}}}{\mathfrak{C}}$
AE	*AB*	*EE*	*EB*
$\dfrac{\mathfrak{F}\mathfrak{a}}{\forall \mathfrak{x} \mathfrak{F}\mathfrak{x}}$	$\dfrac{\forall \mathfrak{x} \mathfrak{F}\mathfrak{x}}{\mathfrak{F}\mathfrak{a}}$	$\dfrac{\mathfrak{F}\mathfrak{a}}{\exists \mathfrak{x} \mathfrak{F}\mathfrak{x}}$	$\dfrac{\exists \mathfrak{x} \mathfrak{F}\mathfrak{x} \quad \overset{[\mathfrak{F}\mathfrak{a}]}{\mathfrak{C}}}{\mathfrak{C}}$
FE	*FB*	*NE*	*NB*
$\dfrac{\overset{[\mathfrak{A}]}{\mathfrak{B}}}{\mathfrak{A} \supset \mathfrak{B}}$	$\dfrac{\mathfrak{A} \quad \mathfrak{A} \supset \mathfrak{B}}{\mathfrak{B}}$	$\dfrac{\overset{[\mathfrak{A}]}{\curlywedge}}{\neg \mathfrak{A}}$	$\dfrac{\mathfrak{A} \quad \neg \mathfrak{A}}{\curlywedge} \qquad \dfrac{\curlywedge}{\mathfrak{D}}$

Gerhard Gentzen (1935) – Natural deduction

$$\frac{\begin{matrix}[A]^x\\ \vdots\\ B\end{matrix}}{A \to B}\,(\to \mathrm{I}^x) \qquad \frac{A \to B \quad A}{B}\,(\to \mathrm{E})$$

$$\frac{A \quad B}{A \mathbin{\&} B}\,(\&\mathrm{I}) \qquad \frac{A \mathbin{\&} B}{A}\,(\&\mathrm{E}_l) \qquad \frac{A \mathbin{\&} B}{B}\,(\&\mathrm{E}_r)$$

Example proof by natural deduction

$$\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})}{(B \mathbin{\&} A) \to (A \mathbin{\&} B)}\,(\to \mathrm{I}^z)$$

Simplifying proofs

$$\dfrac{\dfrac{\begin{matrix}[A]^x\\ \vdots\\ B\end{matrix}}{A \to B}\,(\to \mathrm{I}^x) \qquad \begin{matrix}\vdots\\ A\end{matrix}}{B}\,(\to \mathrm{E}) \qquad \Rightarrow \qquad \begin{matrix}\vdots\\ A\\ \vdots\\ B\end{matrix}$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ A\end{matrix} \quad \begin{matrix}\vdots\\ B\end{matrix}}{A \,\&\, B}\,(\&\mathrm{I})}{A}\,(\&\mathrm{E}_l) \qquad \Rightarrow \qquad \begin{matrix}\vdots\\ A\end{matrix}$$

Simplifying example proof

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})}{(B \mathbin{\&} A) \to (A \mathbin{\&} B)}\,(\to \mathrm{I}^z) \qquad \dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{A \mathbin{\&} B}\,(\to \mathrm{E})$$

Simplifying example proof

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})}{(B \mathbin{\&} A) \to (A \mathbin{\&} B)}\,(\to \mathrm{I}^z) \qquad \dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{A \mathbin{\&} B}\,(\to \mathrm{E})$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})$$

Simplifying example proof

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})}{(B \mathbin{\&} A) \to (A \mathbin{\&} B)}\,(\to \mathrm{I}^z) \qquad \dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{A \mathbin{\&} B}\,(\to \mathrm{E})$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{A}\,(\&\mathrm{E}_r) \qquad \dfrac{\dfrac{[B]^y \quad [A]^x}{B \mathbin{\&} A}\,(\&\mathrm{I})}{B}\,(\&\mathrm{E}_l)}{A \mathbin{\&} B}\,(\&\mathrm{I})$$

$$\Downarrow$$

$$\dfrac{[A]^x \quad [B]^y}{A \mathbin{\&} B}\,(\&\mathrm{I})$$

Alonzo Church (1903–1995)

Alonzo Church (1940) – Typed λ-calculus

$$\dfrac{\begin{array}{c}[x : A]^x \\ \vdots \\ N : B\end{array}}{\lambda x.\, N : A \to B}\;(\to \mathrm{I}^x) \qquad \dfrac{L : A \to B \quad M : A}{L\; M : B}\;(\to \mathrm{E})$$

$$\dfrac{M : A \quad N : B}{(M, N) : A \,\&\, B}\;(\&\mathrm{I}) \qquad \dfrac{L : A \,\&\, B}{\mathrm{fst}\; L : A}\;(\&\mathrm{E}_l) \qquad \dfrac{L : A \,\&\, B}{\mathrm{snd}\; L : B}\;(\&\mathrm{E}_r)$$

Example program: swap

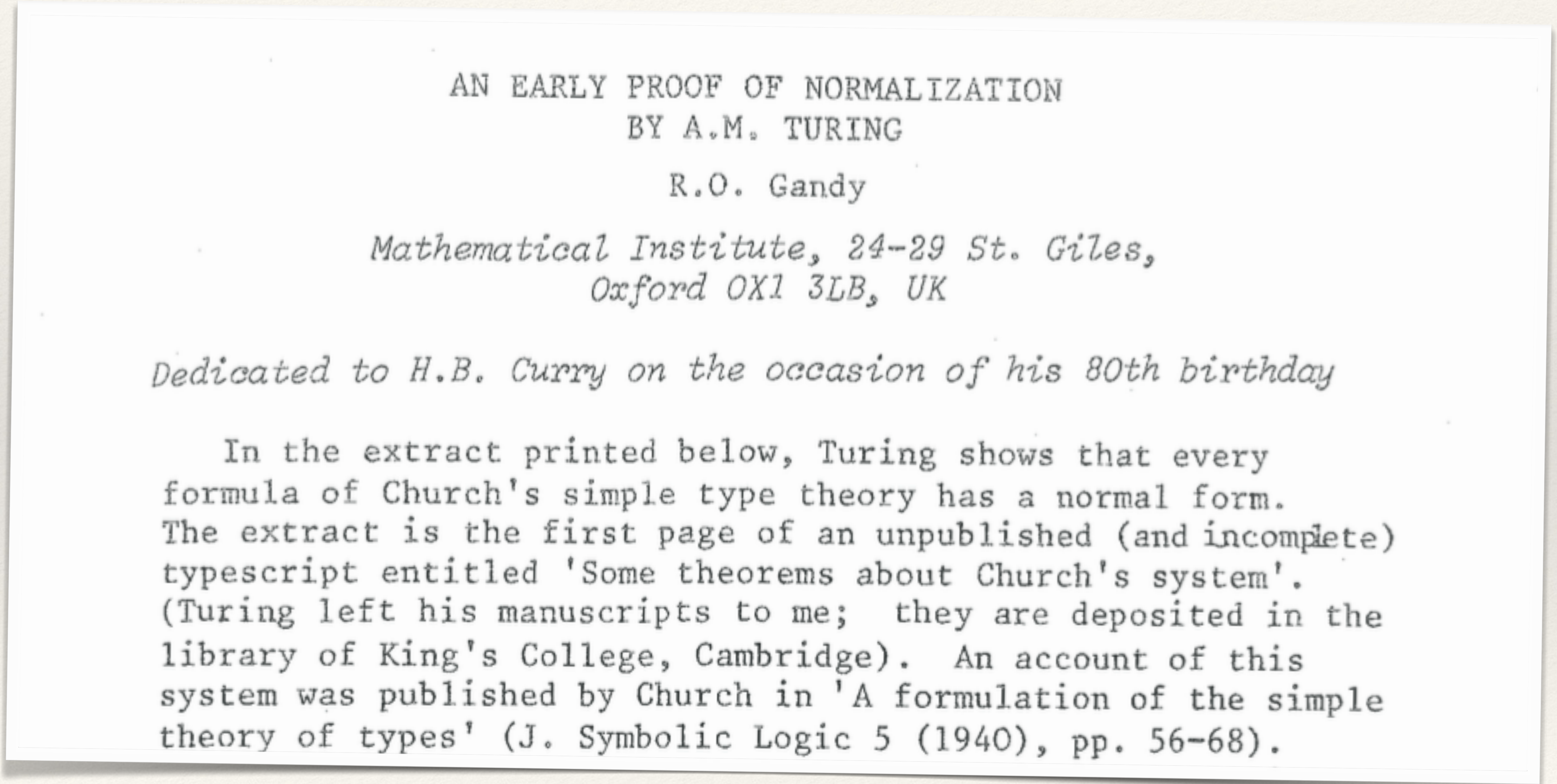
$$\dfrac{\dfrac{\dfrac{[z : B \,\&\, A]^z}{\mathrm{snd}\ z : A}\,(\&\mathrm{E}_r) \qquad \dfrac{[z : B \,\&\, A]^z}{\mathrm{fst}\ z : B}\,(\&\mathrm{E}_l)}{(\mathrm{snd}\ z, \mathrm{fst}\ z) : A \,\&\, B}\,(\&\mathrm{I})}{\lambda z.\ (\mathrm{snd}\ z, \mathrm{fst}\ z) : (B \,\&\, A) \to (A \,\&\, B)}\,(\to \mathrm{I}^z)$$

Evaluating programs

$$\dfrac{\dfrac{\begin{array}{c}[x : A]^x \\ \vdots \\ N : B\end{array}}{\lambda x.\, N : A \to B}\,(\to \mathrm{I}^x) \qquad \begin{array}{c}\vdots \\ M : A\end{array}}{(\lambda x.\, N)\; M : B}\,(\to \mathrm{E}) \qquad \Rightarrow \qquad \begin{array}{c}\vdots \\ M : A \\ \vdots \\ N\{M/x\} : B\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ M : A\end{array} \quad \begin{array}{c}\vdots \\ N : B\end{array}}{(M, N) : A \,\&\, B}\,(\&\mathrm{I})}{\mathrm{fst}\ (M, N) : A}\,(\&\mathrm{E}_l) \qquad \Rightarrow \qquad \begin{array}{c}\vdots \\ M : A\end{array}$$

Alan Turing (1942) – Proof of normalisation



AN EARLY PROOF OF NORMALIZATION
BY A.M. TURING

R.O. Gandy

Mathematical Institute, 24-29 St. Giles,
Oxford OX1 3LB, UK

Dedicated to H.B. Curry on the occasion of his 80th birthday

In the extract printed below, Turing shows that every formula of Church's simple type theory has a normal form. The extract is the first page of an unpublished (and incomplete) typescript entitled 'Some theorems about Church's system'. (Turing left his manuscripts to me; they are deposited in the library of King's College, Cambridge). An account of this system was published by Church in 'A formulation of the simple theory of types' (J. Symbolic Logic 5 (1940), pp. 56-68).

Evaluating example program

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{[z : B \,\&\, A]^z}{\text{snd } z : A}\,(\&\mathrm{E}_r)
      \qquad
      \dfrac{[z : B \,\&\, A]^z}{\text{fst } z : B}\,(\&\mathrm{E}_l)
    }{(\text{snd } z, \text{fst } z) : A \,\&\, B}\,(\&\mathrm{I})
  }{\lambda z.\ (\text{snd } z, \text{fst } z) : (B \,\&\, A) \to (A \,\&\, B)}\,(\to \mathrm{I}^z)
  \qquad
  \dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \,\&\, A}\,(\&\mathrm{I})
}{(\lambda z.\ (\text{snd } z, \text{fst } z))\ (y, x) : A \,\&\, B}\,(\to \mathrm{E})
$$

Evaluating example program

$$\dfrac{\dfrac{\dfrac{\dfrac{[z : B \mathbin{\&} A]^z}{\mathrm{snd}\ z : A}\,(\&\mathrm{E}_r) \qquad \dfrac{[z : B \mathbin{\&} A]^z}{\mathrm{fst}\ z : B}\,(\&\mathrm{E}_l)}{(\mathrm{snd}\ z, \mathrm{fst}\ z) : A \mathbin{\&} B}\,(\&\mathrm{I})}{\lambda z.\ (\mathrm{snd}\ z, \mathrm{fst}\ z) : (B \mathbin{\&} A) \to (A \mathbin{\&} B)}\,(\to \mathrm{I}^z) \qquad \dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \mathbin{\&} A}\,(\&\mathrm{I})}{(\lambda z.\ (\mathrm{snd}\ z, \mathrm{fst}\ z))\ (y, x) : A \mathbin{\&} B}\,(\to \mathrm{E})$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \mathbin{\&} A}\,(\&\mathrm{I})}{\mathrm{snd}\ (y, x) : A}\,(\&\mathrm{E}_r) \qquad \dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \mathbin{\&} A}\,(\&\mathrm{I})}{\mathrm{fst}\ (y, x) : B}\,(\&\mathrm{E}_l)}{(\mathrm{snd}\ (y, x), \mathrm{fst}\ (y, x)) : A \mathbin{\&} B}\,(\&\mathrm{I})$$

Evaluating example program

$$\dfrac{\dfrac{\dfrac{\dfrac{[z : B \,\&\, A]^z}{\text{snd } z : A}\,(\&\mathrm{E}_r) \qquad \dfrac{[z : B \,\&\, A]^z}{\text{fst } z : B}\,(\&\mathrm{E}_l)}{(\text{snd } z, \text{fst } z) : A \,\&\, B}\,(\&\mathrm{I})}{\lambda z.\,(\text{snd } z, \text{fst } z) : (B \,\&\, A) \to (A \,\&\, B)}\,(\to \mathrm{I}^z) \qquad \dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \,\&\, A}\,(\&\mathrm{I})}{(\lambda z.\,(\text{snd } z, \text{fst } z))\;(y, x) : A \,\&\, B}\,(\to \mathrm{E})$$

$$\Downarrow$$

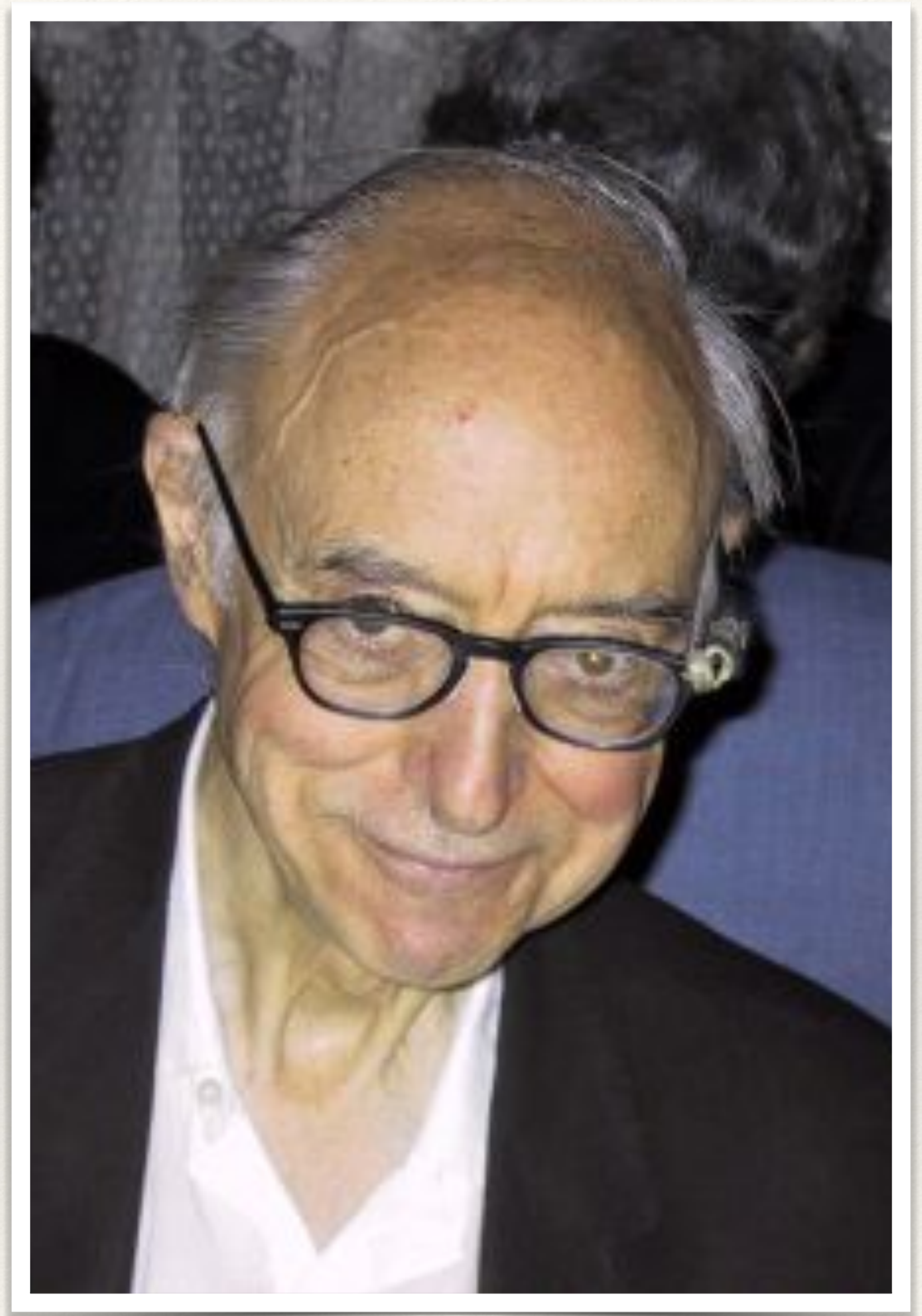
$$\dfrac{\dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \,\&\, A}\,(\&\mathrm{I})}{\text{snd } (y, x) : A}\,(\&\mathrm{E}_r) \qquad \dfrac{\dfrac{[y : B]^y \quad [x : A]^x}{(y, x) : B \,\&\, A}\,(\&\mathrm{I})}{\text{fst } (y, x) : B}\,(\&\mathrm{E}_l)}{(\text{snd } (y, x), \text{fst } (y, x)) : A \,\&\, B}\,(\&\mathrm{I})$$

$$\Downarrow$$

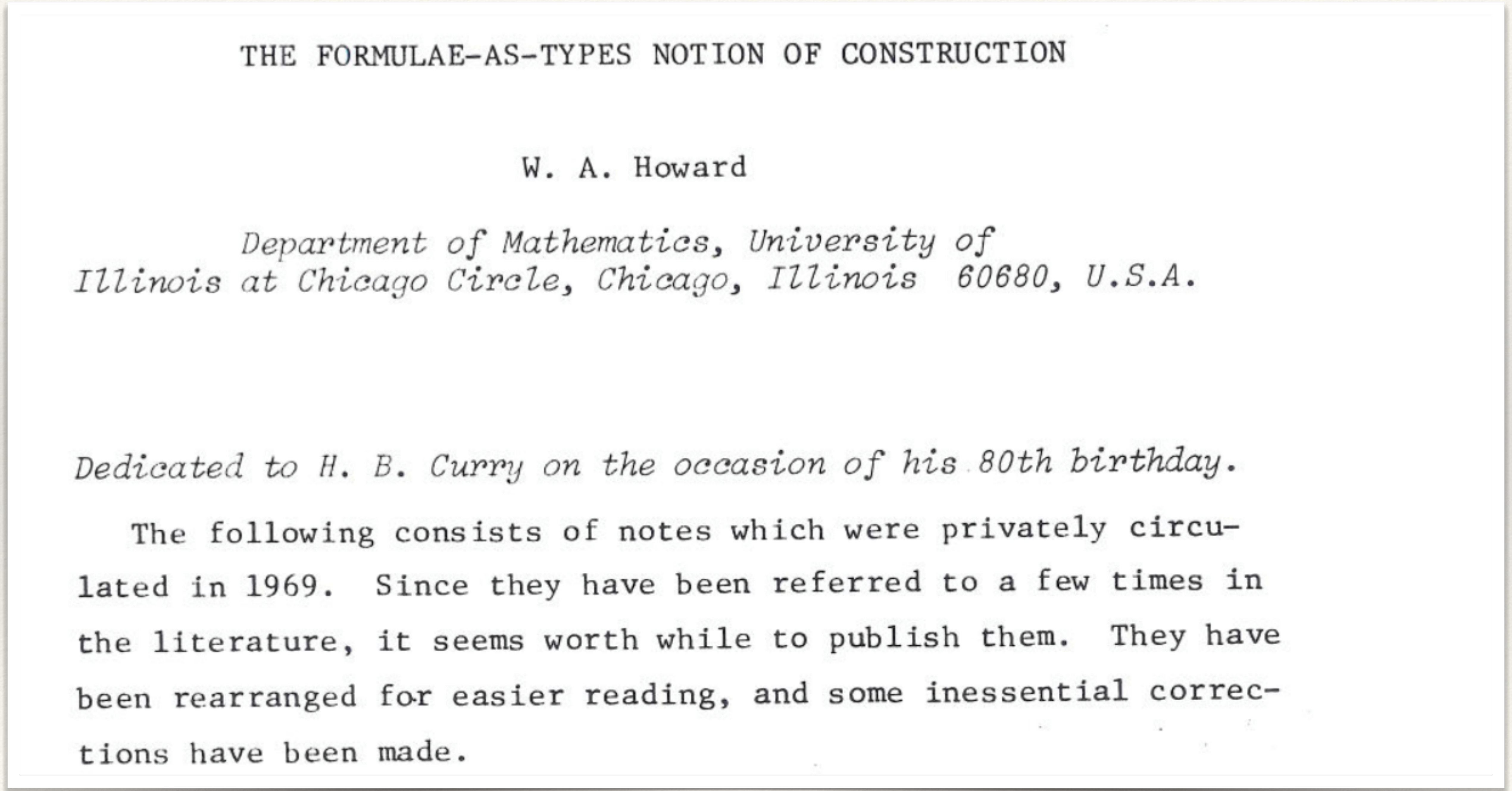
$$\dfrac{[x : A]^x \quad [y : B]^y}{(x, y) : A \,\&\, B}\,(\&\mathrm{I})$$

Haskell Curry (1900–1982)

William Howard (1926–)



Howard (1980) – Propositions as Types



THE FORMULAE-AS-TYPES NOTION OF CONSTRUCTION

W. A. Howard

Department of Mathematics, University of Illinois at Chicago Circle, Chicago, Illinois 60680, U.S.A.

Dedicated to H. B. Curry on the occasion of his 80th birthday.

The following consists of notes which were privately circulated in 1969. Since they have been referred to a few times in the literature, it seems worth while to publish them. They have been rearranged for easier reading, and some inessential corrections have been made.

Curry-Howard correspondence

propositions	*as*	types
proofs	*as*	programs
proof simplification	*as*	program evaluation

Curry-Howard correspondence

Natural Deduction Gentzen (1935)	$\Leftrightarrow$	**Typed λ-calculus** Church (1940)
Type schemes Hindley (1969)	$\Leftrightarrow$	**ML Type System** Milner (1975)
System F Girard (1972)	$\Leftrightarrow$	**Polymorphic λ-calculus** Reynolds (1974)
Modal Logic Lewis (1910)	$\Leftrightarrow$	**Monads (state, exceptions)** Kleisli (1965), Moggi (1987)
Double-negation translation Gödel (1932), Gentzen(1935)	$\Leftrightarrow$	**Continuations** Reynolds (1972)

Functional programming languages

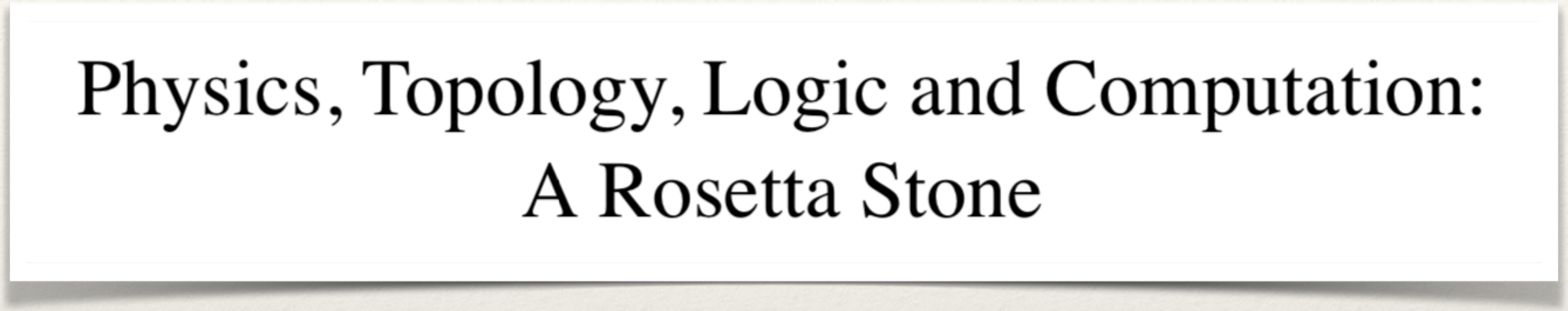
- Lisp (McCarthy, 1958)
- Scheme (Steele and Sussman, 1970)
- ML (Milner et al., 1973)
- Haskell (Hudak, Peyton Johnes, Wadler, 1987)
- Erlang (Armstrong, Virding, Williams, 1987)
- OCaml (Leroy, 1996)
- Scala (Odersky, 2003)
- F# (Syme, 2005)
- Idris (Brady, 2009)

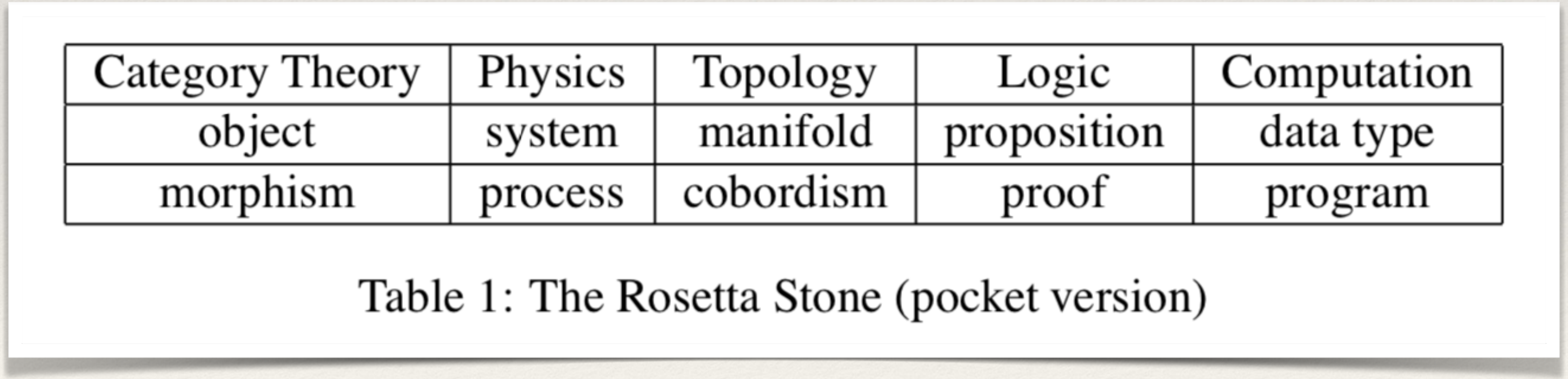
Theorem provers

- Automath (de Bruijn, 1970)
- ML/LCF (Milner et al., 1973)
- Type Theory (Per Martin-Löf, 1975)
- Mizar (Trybulec, 1975)
- NuPrl (Constable, 1985)
- HOL (Gordon and Melham, 1988)
- Coq (Huet and Coquend, 1988)
- Isabelle (Paulson, 1993)
- Epigram (McBride and McKinna, 2004)
- Agda (Norell, 2005)

Extras

Baez, Stay (2009) – A Rosetta Stone

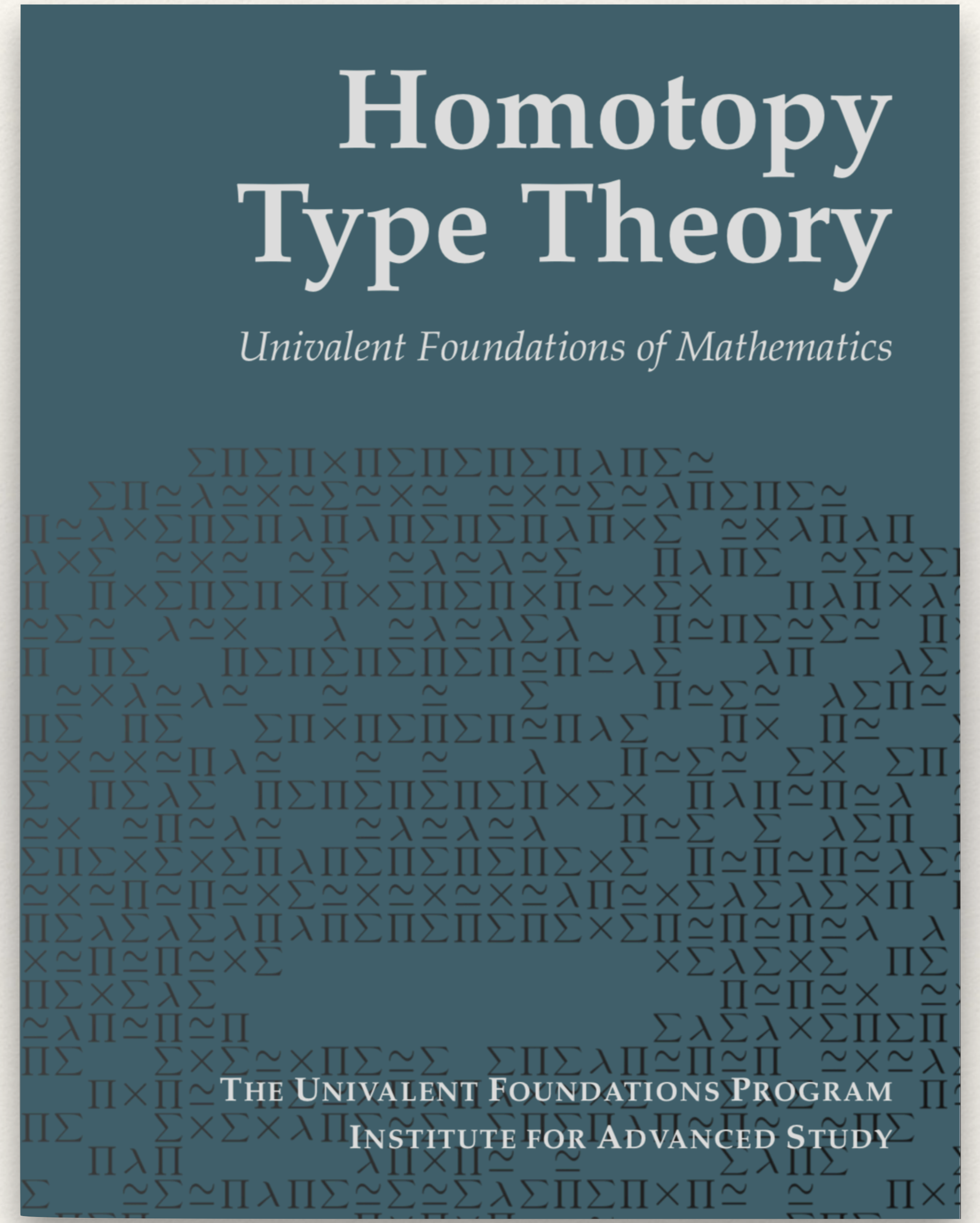

Physics, Topology, Logic and Computation:
A Rosetta Stone



Category Theory	Physics	Topology	Logic	Computation
object	system	manifold	proposition	data type
morphism	process	cobordism	proof	program

Table 1: The Rosetta Stone (pocket version)

Homotopy Type Theory (2013)



Homotopy Type Theory

Univalent Foundations of Mathematics

The Univalent Foundations Program
Institute for Advanced Study

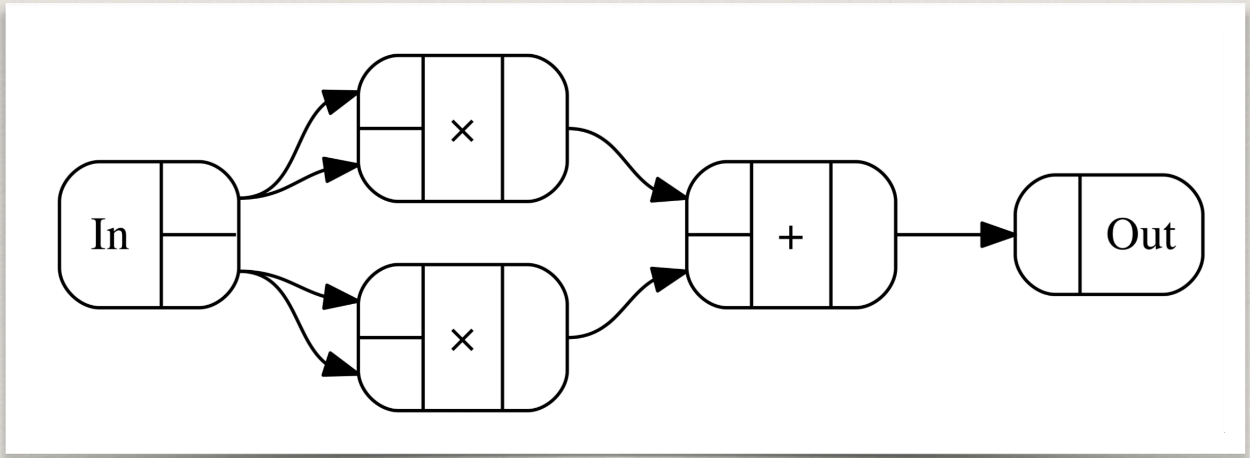
Elliott (2017) – Compiling to Categories

Compiling to Categories

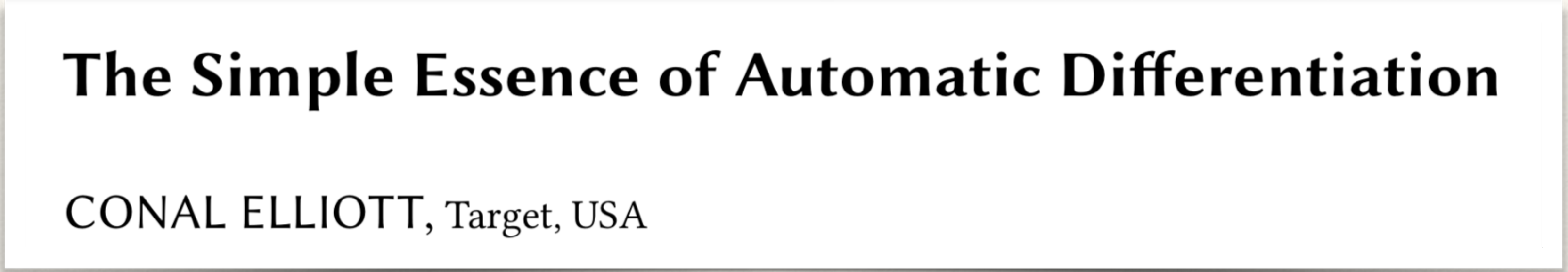
CONAL ELLIOTT, Target, USA

$$magSqr\ (a, b) = sqr\ a + sqr\ b$$

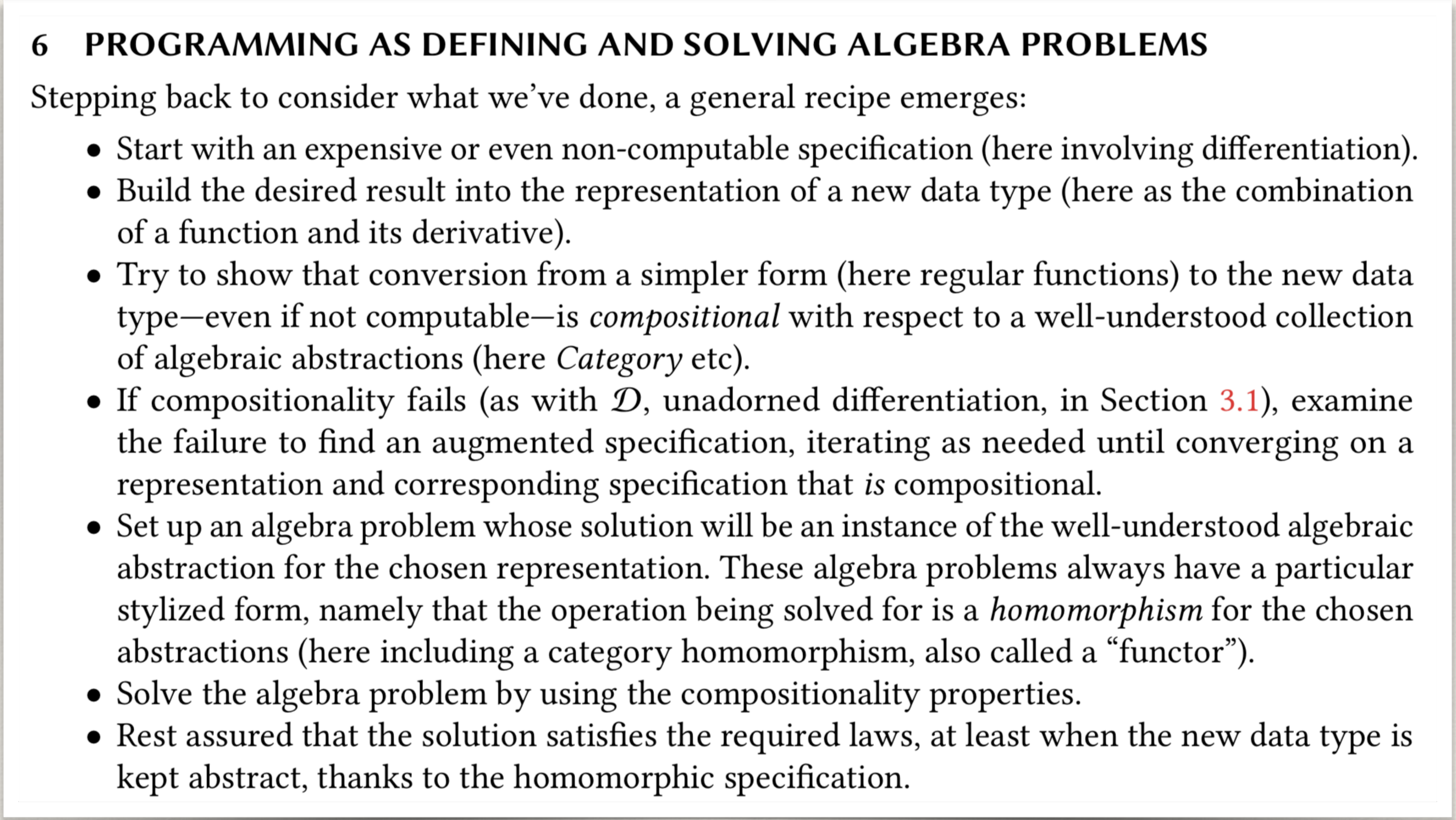
$$magSqr = addC \circ (mulC \circ (exl \vartriangle exl) \vartriangle mulC \circ (exr \vartriangle exr))$$



Elliott (2018)
The Simple Essence of Automatic Differentiation


The Simple Essence of Automatic Differentiation

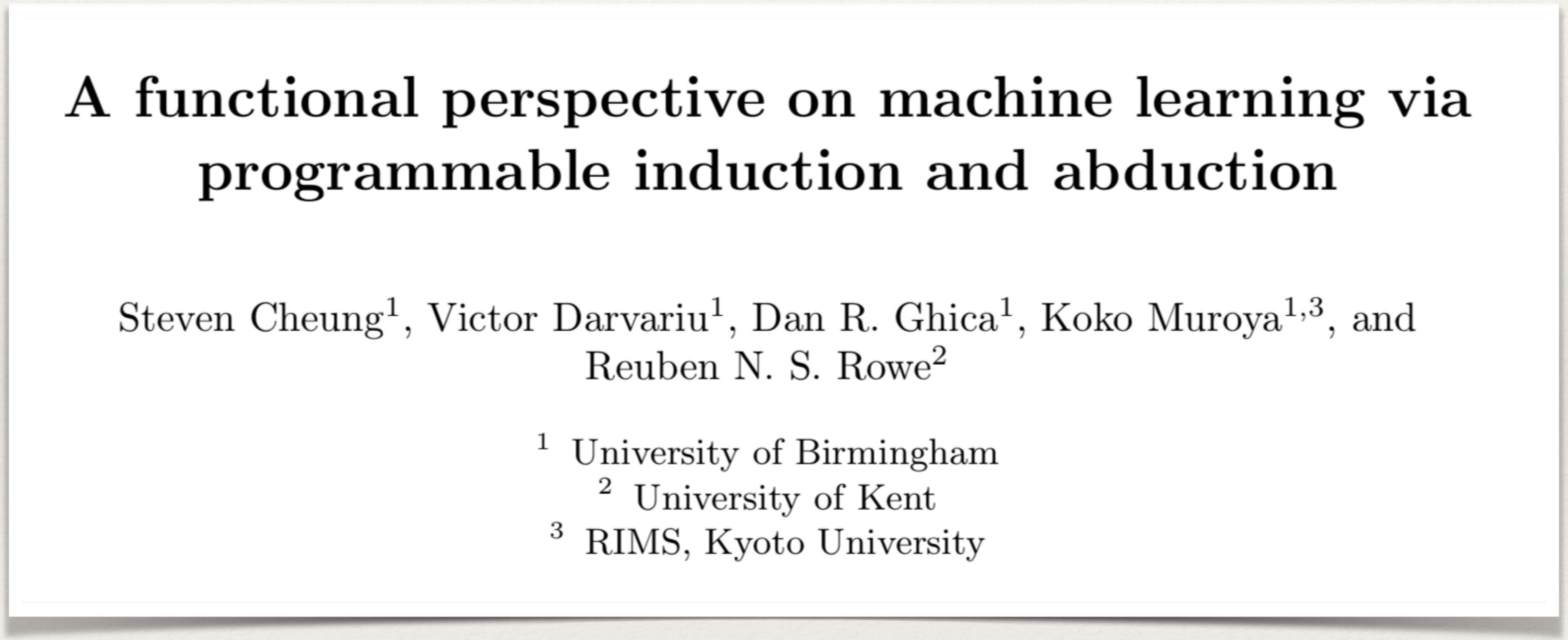
CONAL ELLIOTT, Target, USA


6 PROGRAMMING AS DEFINING AND SOLVING ALGEBRA PROBLEMS

Stepping back to consider what we've done, a general recipe emerges:

- Start with an expensive or even non-computable specification (here involving differentiation).
- Build the desired result into the representation of a new data type (here as the combination of a function and its derivative).
- Try to show that conversion from a simpler form (here regular functions) to the new data type—even if not computable—is *compositional* with respect to a well-understood collection of algebraic abstractions (here *Category* etc).
- If compositionality fails (as with $\mathcal{D}$, unadorned differentiation, in Section 3.1), examine the failure to find an augmented specification, iterating as needed until converging on a representation and corresponding specification that *is* compositional.
- Set up an algebra problem whose solution will be an instance of the well-understood algebraic abstraction for the chosen representation. These algebra problems always have a particular stylized form, namely that the operation being solved for is a *homomorphism* for the chosen abstractions (here including a category homomorphism, also called a "functor").
- Solve the algebra problem by using the compositionality properties.
- Rest assured that the solution satisfies the required laws, at least when the new data type is kept abstract, thanks to the homomorphic specification.

Cheung et al. (2018)
A principled functional language for machine learning



A functional perspective on machine learning via programmable induction and abduction

Steven Cheung[1], Victor Darvariu[1], Dan R. Ghica[1], Koko Muroya[1,3], and Reuben N. S. Rowe[2]

[1] University of Birmingham
[2] University of Kent
[3] RIMS, Kyoto University