Entscheidungsproblem!

July 2022 – JSIP — Hao Lian

$$\underbrace{\underbrace{\text{Ent} \cdot \overbrace{\text{scheiden}}^{\text{separate}}}_{\text{decide}} \cdot \text{ungs}}_{\text{decision}} \cdot \text{problem}$$

The *Principia Mathematica* (1910-1913) is metal

$$\ast 54\cdot 43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$$

Dem.

$$\vdash . \ast 54\cdot 26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$$

$$[\ast 51\cdot 231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$$

$$[\ast 13\cdot 12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$$

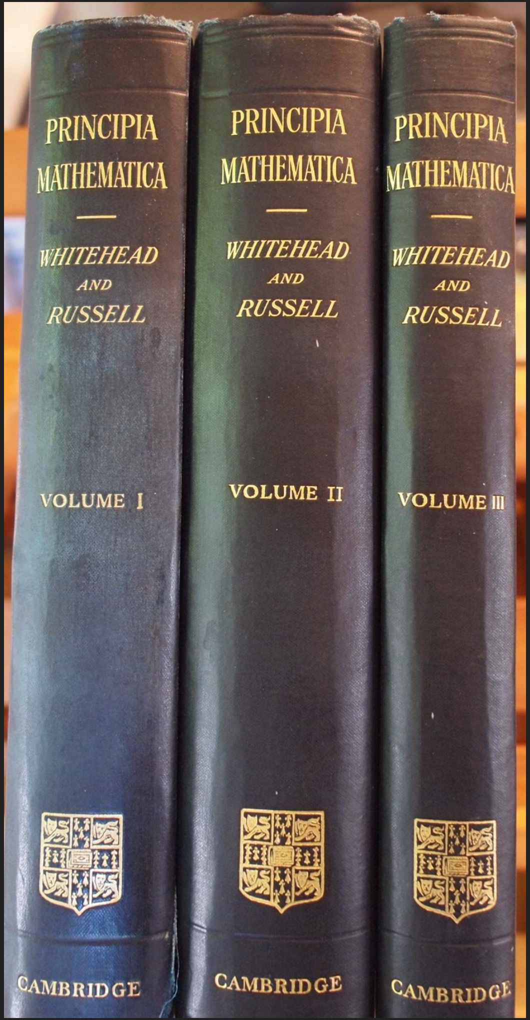
$$\vdash . (1) . \ast 11\cdot 11\cdot 35 . \supset$$

$$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$$

$$\vdash . (2) . \ast 11\cdot 54 . \ast 52\cdot 1 . \supset \vdash . \text{Prop}$$

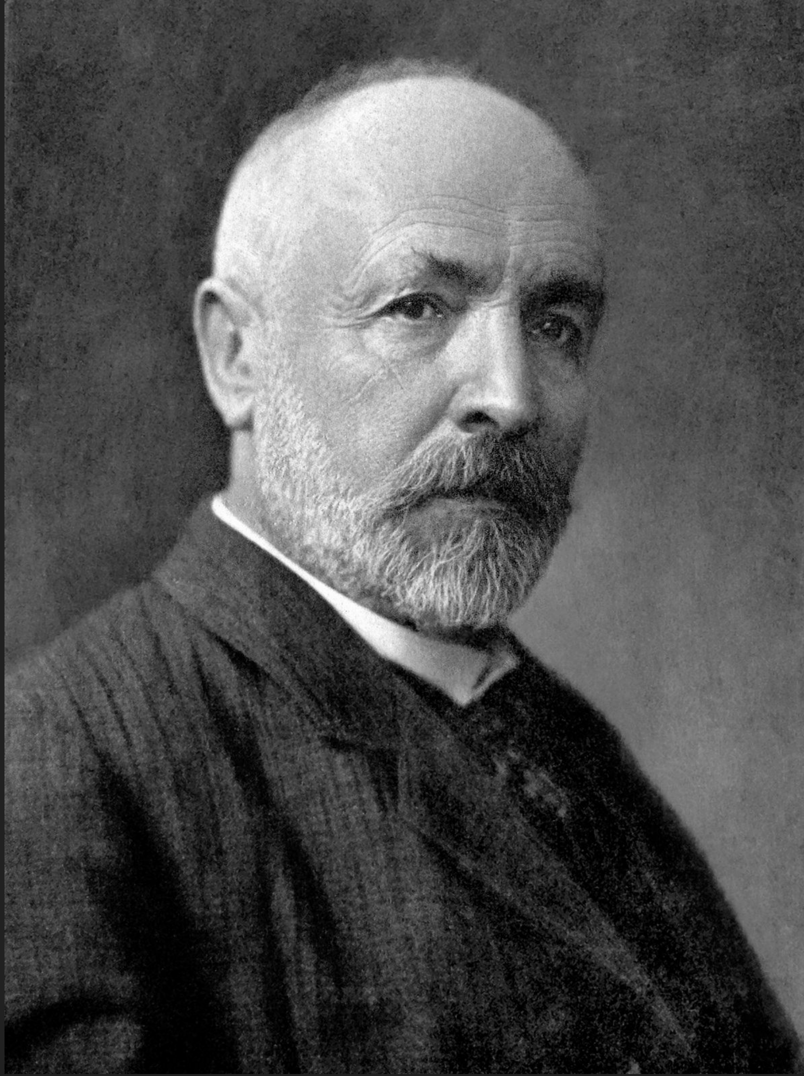
From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

"1+1=2," The Universe of Discourse


PRINCIPIA MATHEMATICA
WHITEHEAD AND RUSSELL
VOLUME I
CAMBRIDGE
PRINCIPIA MATHEMATICA
WHITEHEAD AND RUSSELL
VOLUME II
CAMBRIDGE
PRINCIPIA MATHEMATICA
WHITEHEAD AND RUSSELL
VOLUME III
CAMBRIDGE

Mathematics in 1910

- The **professionalization** of mathematics
- 1874-1884: Georg Cantor's set theory
  - **POWER SET** and **1-to-1 CORRESPONDENCE**
  - Not all infinities are created equal!
  - Continuum Hypothesis [Hilbert's first problem]
- Is God real?
  - Infinity is the proof of God (Cantor)
  - Infinity is a blasphemy upon God (neo-Thomists)



You cannot prove $1 + 1 = 2$ *and* $1 + 1 \neq 2$.

Arithmetic is consistent

A set of axioms is **consistent** iff there is no formula *f* such that you can prove both *f* and *!f*.

Arithmetic is consistent

Right? … right?

Kurt Gödel
(1906-1978)

- Gödel axiomatizes first-order logic (1930 Ph. D. thesis)
  - A formula is provable from the axioms iff it is *valid*
  - A formula is *effectively computable* iff a "mechanical procedure" can enumerate every valid statement in first-order logic
  - Given a formula *f*, though, this is not enough to determine if *f* is valid mechanically: what if the program never halts?
- Gödel then proves no such complete and computable axiomatization of arithmetic exists
  - By doing two things: **turning propositions into numbers** and **using self-referential propositions**
  - But his proof has a hole: what *is* computable?

What is an **algorithm**?

But what is an **algorithm**?

- Church (1933-1936) *
- Turing (1936)

* Gödel (1934)

Alonzo Church (1903-1995)

Untyped lambda calculus

- **Variable:** x in **Λ**.
- **Abstraction:**
  If y in **Λ**, then $\underline{(\text{fun } x \to y)}$ also in **Λ**.
- **Application:**
  If x, y in **Λ**, then $\underline{x\ y}$ also in **Λ**.

Untyped lambda calculus

- **Rules of inference**: alpha- and beta-reduction
- $\underline{(\text{fun } x \to L)\ y} \equiv \underline{L\ [x := y]}$

Untyped lambda calculus is as strong as arithmetic. *How?*

Encoding Booleans in the λ-Calculus

B	λ-calculus
`true`	$\lambda x.\ \lambda y.\ x$
`false`	$\lambda x.\ \lambda y.\ y$
`&&`	$\lambda x.\ \lambda y.\ ((x\ y)\ \texttt{false})$
`||`	$\lambda x.\ \lambda y.\ ((x\ \texttt{true})\ y)$
`!`	$\lambda x.\ ((x\ \texttt{false})\ \texttt{true})$
`if`	$\lambda c.\ \lambda t.\ \lambda e.\ ((c\ t)\ e)$

for and while loops are possible too!

This is known as the *Church encoding of Booleans*, or simply *Church Booleans*.

Example:

$$(\texttt{true}\ \texttt{\&\&}\ \texttt{false})$$

$$\equiv \quad (\lambda x.\ \lambda y.\ ((x\ y)\ \texttt{false}))\ (\lambda x.\ \lambda y.\ x)\ (\lambda x.\ \lambda y.\ y)$$

$$\to \quad (\lambda y.\ (((\lambda x.\ \lambda y.\ x)\ y)\ \texttt{false}))\ (\lambda x.\ \lambda y.\ y)$$

$$\to \quad (\ ((\lambda x.\ \lambda y.\ x)\ (\lambda x.\ \lambda y.\ y))\ \texttt{false})$$

$$\to \quad (\ (\lambda y.\ (\lambda x.\ \lambda y.\ y))\ \texttt{false})$$

$$\to \quad (\lambda x.\ \lambda y.\ y)$$

$$\equiv \quad \texttt{false}$$


Alan Turing (1912-1954)

Alan Turing (1912-1954)

Alan Turing (1912-1954)

Alan Turing (1912-1954)

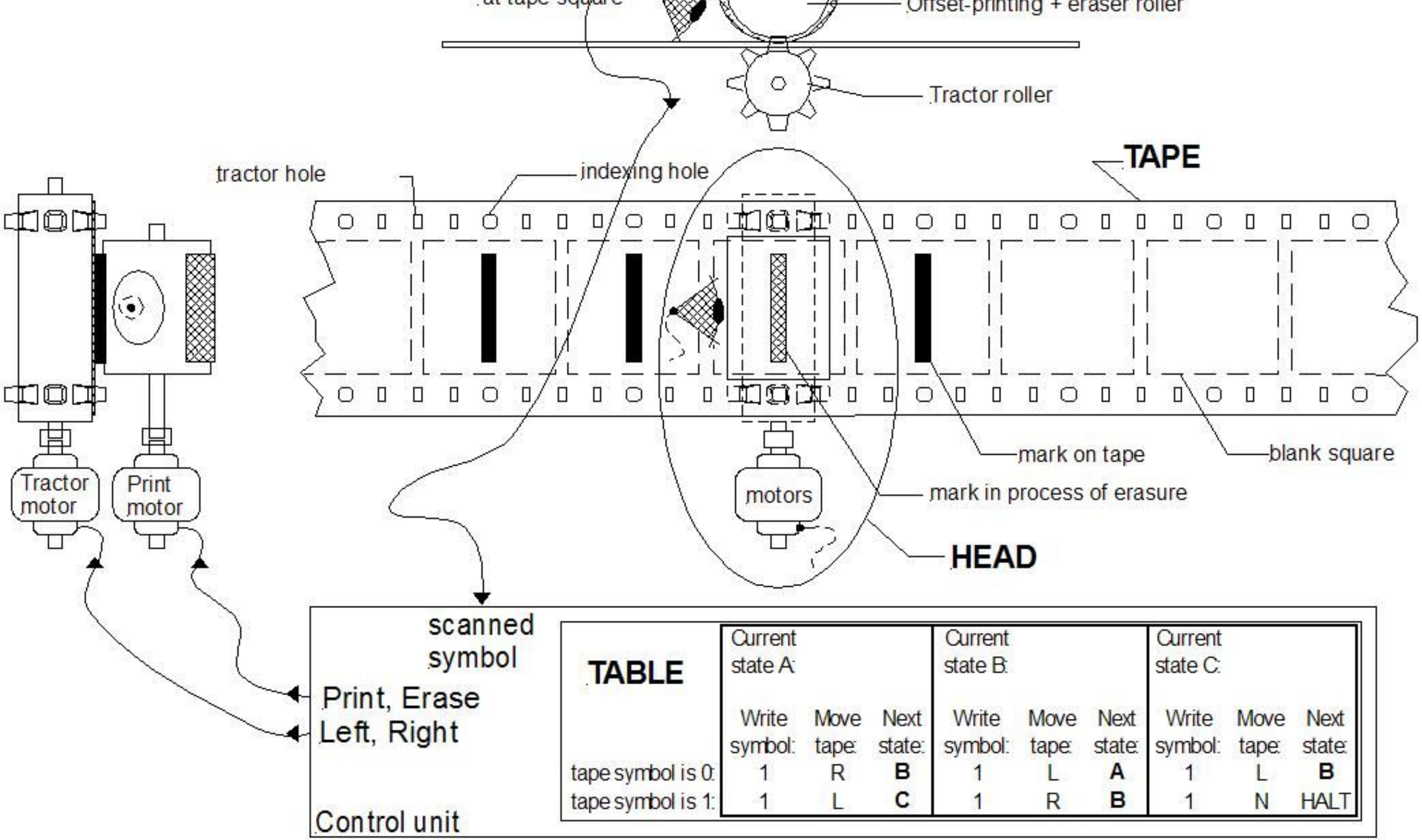


TABLE	Current state A:			Current state B:			Current state C:		
	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	**B**	1	L	**A**	1	L	**B**
tape symbol is 1:	1	L	**C**	1	R	**B**	1	N	HALT

A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Converting propositions to numbers

- Not all real numbers are computable
- A computable function is just a computable number in disguise: 0.14159… ⇔ $f(0) = 1; f(1) = 4; f(2) = 1; f(3) = 5; f(4) = 9$

Self-referentiality

An *universal Turing machine* is a Turing machine capable of simulating an arbitrary Turing machine on an arbitrary input. (Stored-program computers)

Turing-complete

Any system is said to be Turing-complete iff it can simulate a Turing machine

What else is Turing-complete?

- C++ templates (oops)
- TypeScript type system (oops?)
- x86's [mov] instruction
- Magic: The Gathering
- Pokémon Yellow
- Bits and pieces of most executable binaries (yikes! — return-oriented programming attack)

What *isn't* Turing-complete?

- Regular expressions
- Simply-typed lambda calculus
- Well-constructed type systems
- Not... much else?

Entscheidungsproblem!

Is there an algorithm to **decide** *every* statement as DERIVABLE or UNDERIVABLE?

(Hilbert and Ackermann, 1928)

David Hilbert (1862-1943)

The halting problem (a sketch)

- Does an algorithm exist that, given a computer program and some input, can determine whether the program will finish running or continue to run forever?
- No!
  - The proof relies on **converting propositions to numbers** and **self-reference**
- What is an algorithm?
  - A Turing machine!
- The halting problem is *not computable*
- *Therefore, there cannot exist an algorithm to decide every statement as derivable?*

Entscheidungsproblem!

- Turing machines: NO

Entscheidungsproblem!

- Turing machines: NO
- Church's lambda calculus: NO

Lambda calculus ⇔ Turing machines

- Lambda calculus is Turing-complete
- Turing-machines are Turing-complete (Universal Turing Machine)

Definition (Church-Turing Thesis)
A function f on natural numbers is *computable* iff f is λ-computable iff f is Turing-computable (Rosser 1939)

But what is an **algorithm**?

- Church (1933-1936) *
- Turing (1936)

* Gödel (1934)

"We had not perceived the sharp concept of mechanical procedures sharply before Turing, who brought us to the right perspective."

— Gödel (Wang 1974)

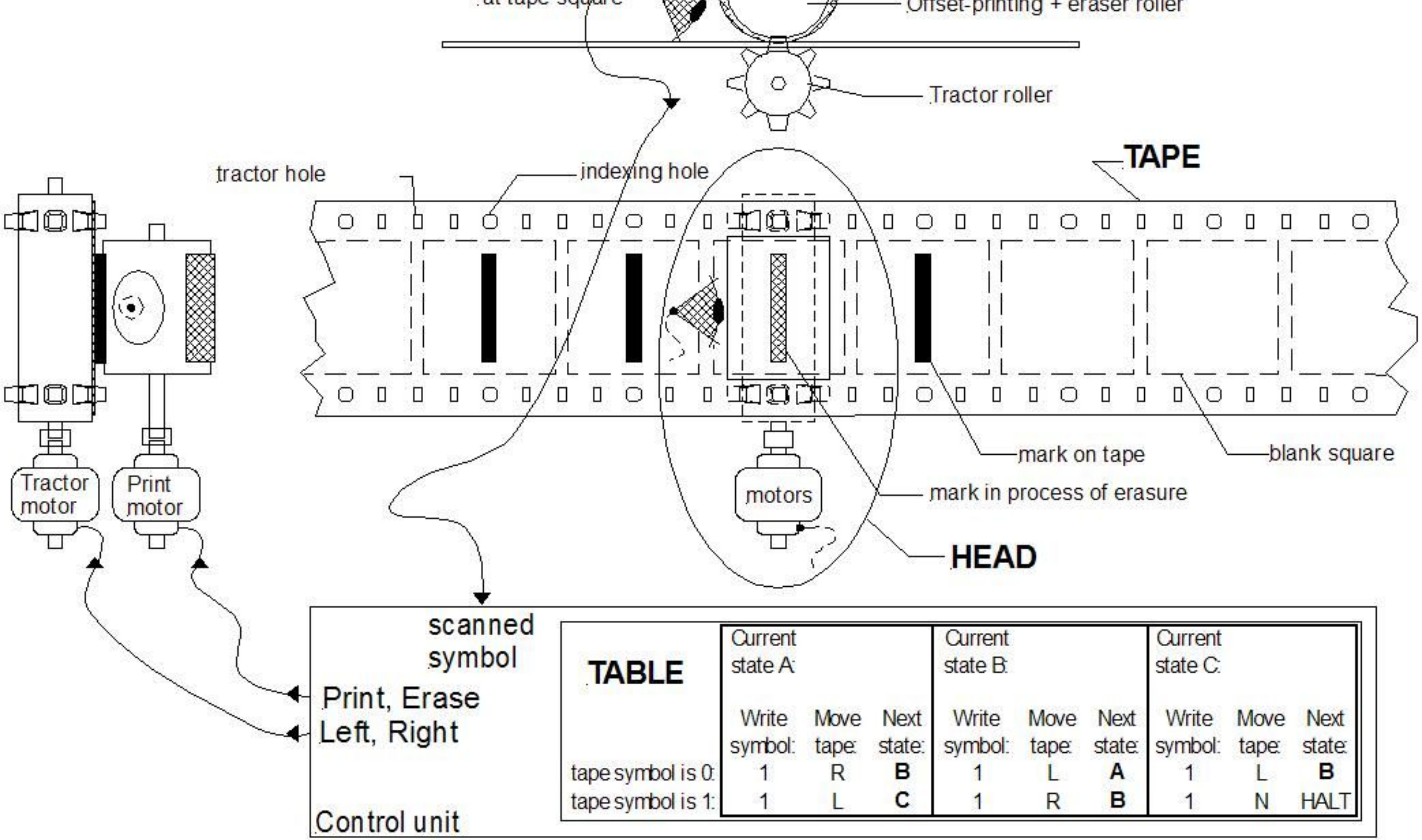


TABLE	Current state A:			Current state B:			Current state C:		
	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:	Write symbol:	Move tape:	Next state:
tape symbol is 0:	1	R	**B**	1	L	**A**	1	L	**B**
tape symbol is 1:	1	L	**C**	1	R	**B**	1	N	HALT

A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

“ not perceived the sharp
co chanical
pr before Turing,
who b
perspective.

— Go

Gödel would have been
a C++ programmer

Alan Turing’s Resume

- 1936, age 24 (senior in college): Turing machines; Entscheidungsproblem answered as “No”; halting problem is undecidable
- 1940: General decryption of Nazi Germany’s Enigma cipher machine
- 1941: proposes marriage to Joan “unfazed” Clarke
- 1946: first comprehensive stored-program computer design
- 1948: Turbochamp (chess engine) begins
- 1950: Turing test
- 1951: mathematical biological model of morphogenesis
- 1952: convicted of indecency
- 1954, age 42: death

My dear Norman,

I don't think I really do know much about jobs, except the one I had during the war, and that certainly did not involve any travelling. I think they do take on conscripts. It certainly involved a good deal of hard thinking, but whether you'd be interested I don't know. Philip Hall was in the same racket and on the whole, I should say, he didn't care for it. However I am not at present in a state in which I am able to concentrate well, for reasons explained in the next paragraph.

I've now got myself into the kind of trouble that I have always considered to be quite a possibility for me, though I have usually rated it at about 10:1 against. I shall shortly be pleading guilty to a charge of sexual offences with a young man. The story of how it all came to be found out is a long and fascinating one, which I shall have to make into a short story one day, but haven't the time to tell you now. No doubt I shall emerge from it all a different man, but quite who I've not found out.

Glad you enjoyed broadcast. Jefferson certainly was rather disappointing though. **I'm afraid that the following syllogism may be used by some in the future.**

Turing believes machines think
Turing lies with men
Therefore machines do not think

Yours in distress,
Alan

Alan Turing to Norman Routlege (1952)

Where to from here?

- Simply typed lambda calculus + Hindley-Milner type inference = functional programming
- Nondeterministic Turing machines (**P ?= NP**)
- Can humans replicate human intelligence?
- *Should* humans replicate human intelligence?
- What can we make with computers?
- Should code be free?

- 1937: What *is* a computer?
- 1948: 1st stored-program computer
- 1969: Unix 1st edition
- 1972: C programming language
- 1973: ML programming language
- 1983: 1st computer with GUI (Apple Lisa)


profile
READY

Can he end the talk with a takeaway?

Down with fascists

Don't embarrass God by trying to understand the nature of infinity

Takeaway: Computer science is a young field

Software engineering doubly so

All the things that can be discovered have *not* yet been discovered

Go out and make things