XIV

AUTOMATA & FORMAL LANGUAGES
Fourteenth lecture
Saturday, 5 November 2022

	regular (type 3)	context-free (type 2)
Closure properties.		
Concatenation	✓	✓
Union	✓	✓
Intersection	✓	×
Complementation	✓	×
Difference	✓	×
Decision problems.		
Word problem	✓	✓
Emptiness problem	✓	✓
Equivalence problem	✓	×

SUMMARY FROM LECTURE XIII

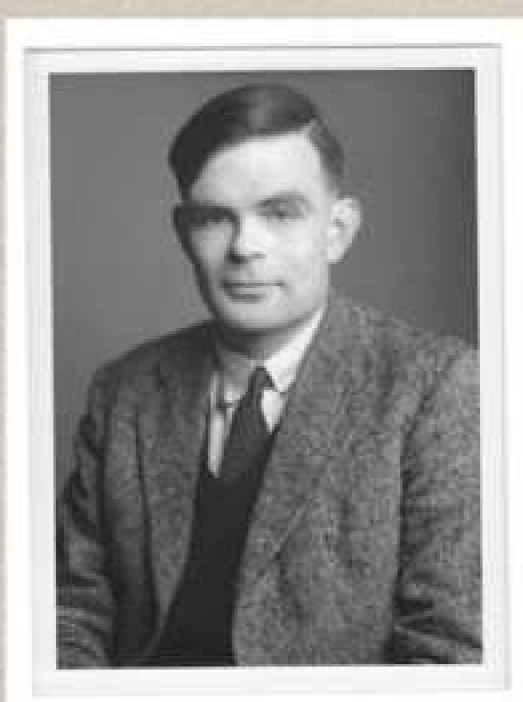
how do we prove that the equivalence problem for context-free languages is not solvable?

$\Longleftrightarrow$

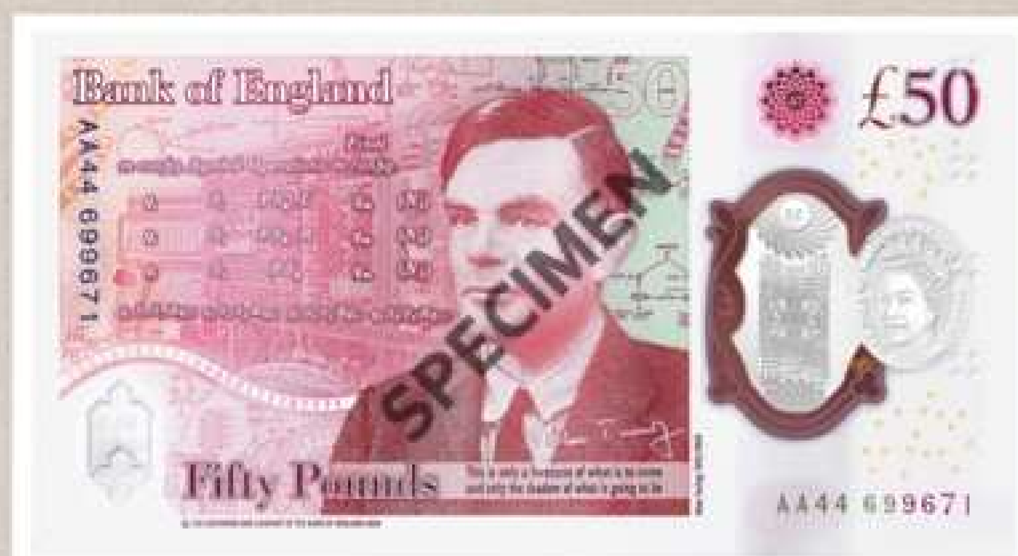
... that there is no algorithm that determines whether $\mathcal{L}(G) = \mathcal{L}(G')$?

$\longrightarrow$ We need a formal definition of **ALGORITHM** !

Chapter 4: COMPUTABILITY



Alan TURING
1912-1954
King's College 1931-1934
Fellow at King's 1935



Proceedings of the London Mathematical Society 1936

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

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

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. Although the subject of this paper is ostensibly the computable *numbers*. it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

It is my contention that these operations include all those which are used in the computation of a number. The defence of this contention will be easier when the theory of the machines is familiar to the reader. In the next section I therefore proceed with the development of the theory and assume that it is understood what is meant by "machine", "tape", "scanned", etc.

2. *Definitions.*

Automatic machines.

If at each stage the motion of a machine (in the sense of § 1) is *completely* determined by the configuration, we shall call the machine an "automatic machine" (or *a*-machine).

REGISTER MACHINE

A register machine works over an alphabet Σ, has finitely many states and finitely many **REGISTERS**

These are LIFO (last-in-first-out) storage units, containing a word $w \in W$ and the machine is able to access the last letter, remove it, add a new one or leave it alone.

CONFIGURATION OF LENGTH $n+1$

$$(q, w_0, \dots, w_n) \in Q \times W^{n+1}$$

called a **CONFIGURATION** or **SNAPSHOT** of a computation

Our transition functions should be:

$$\delta : Q \times W^{n+1} \longrightarrow Q \times W^{n+1}$$

Note that there are uncountably many of these functions, so we need to be more restrictive.

(Σ, Q) - instructions

—

Let Σ be an alphabet and Q a non-empty finite set whose elements we shall call *states*. A tuple of the form

$$
\begin{aligned}
(0, k, a, q) &\in \mathbb{N} \times \mathbb{N} \times \Sigma \times Q, \\
(1, k, a, q, q') &\in \mathbb{N} \times \mathbb{N} \times \Sigma \times Q \times Q, \\
(2, k, q, q') &\in \mathbb{N} \times \mathbb{N} \times \times Q \times Q \text{ or} \\
(3, k, q, q') &\in \mathbb{N} \times \mathbb{N} \times Q \times Q
\end{aligned}
$$

is called a (Σ, Q)-*instruction*. For improved readability, we write

$$
\begin{aligned}
+(k, a, q) &:= (0, k, a, q), && \text{("add")} \\
?(k, a, q, q') &:= (1, k, a, q, q'), && \text{("check")} \\
?(k, \varepsilon, q, q') &:= (2, k, q, q') \text{ and} && \text{("check")} \\
-(k, q, q') &:= (3, k, q, q') && \text{("remove")}
\end{aligned}
$$

Let $\mathrm{Instr}(\Sigma, Q)$ be the set of (Σ, Q)-instructions.
This is an ∞ set, but finite, if you bound k.

Instruction	Interpretation
$+(k, a, q)$	"Add the letter a to the content of register k and go to state q."
$?(k, a, q, q')$	"Check whether the last letter in register k is a; if so, go to state q; otherwise, go to state q'."
$?(k, \varepsilon, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, go to state q'."
$-(k, q, q')$	"Check whether register k is empty; if so, go to state q; otherwise, remove the final letter of its content and go to state q'".

Definition A tuple $(\Sigma, Q, P) = M$ is called a **Σ-register machine** if Σ is an alphabet, Q is a finite set of STATES with two special states

$$q_S \neq q_H$$

q_S: start state; q_H: halt state

$$P : Q \longrightarrow \mathrm{Instr}(\Sigma, Q).$$

PROGRAM

If $Q = \{q_0, q_1, \ldots, q_n\}$, think of P as

$$\begin{aligned} q_0 &\longmapsto P(q_0) \\ q_1 &\longmapsto P(q_1) \\ &\vdots \\ q_n &\longmapsto P(q_n) \end{aligned}$$

Each of these is called a PROGRAM LINE.

Since Q is finite, only finitely many numbers k show up in $P(q)$ for $q \in Q$. The maximal k showing up is called the UPPER REGISTER INDEX of M.

Def 4.2 If M is a RM with upper reg. index n and $\vec{w} = (w_0, \dots, w_n) \in \mathbb{W}^{n+1}$, we define the COMPUTATION SEQUENCE of M with input $\vec{w}$.

We say that a sequence $C := (q, w_0, ..., w_n) \in Q \times \mathbb{W}^{n+1}$ is a *configuration* or *snapshot* of length $n+1$. In such a configuration, the first entry q is called the *state* of the configuration and the rest is called the *register content* of the configuration. If M is a register machine with upper register index n and C is any configuration of length $m \geq n+1$, then we can define the action of M on C: we say that M *transforms* C *to* C' if the following is true:

Case 1. If $P(q) = +(k, a, q')$ and $C' = (q', w_0, ..., w_{k-1}, w_k a, w_{k+1}, ..., w_m)$.

Case 2. If $P(q) = ?(k, a, q', q'')$,

Subcase 2a. $w_k = wa$ for some w and $C' = (q', w_0, ..., w_m)$ or

Subcase 2b. $w_k \neq wa$ for any w and $C' = (q'', w_0, ..., w_m)$.

Case 3. If $P(q) = ?(k, \varepsilon, q', q'')$,

Subcase 3a. $w_k = \varepsilon$ and $C' = (q', w_0, ..., w_m)$ or

Subcase 3b. $w_k \neq \varepsilon$ and $C' = (q'', w_0, ..., w_m)$.

Case 4. If $P(q) = -(k, q', q'')$,

Subcase 4a. $w_k = \varepsilon$ and $C' = (q', w_0, ..., w_m)$ or

Subcase 4b. $w_k = wa$ for some a and $C' = (q'', w_0, ..., w_{k-1}, w, w_{k+1}, ..., w_m)$.

This defines a notion of

M transforms C to C'.

Define the computation sequence by

$$C(0, M, \vec{w}) = (q_s, \vec{w})$$

$$C(k+1, M, \vec{w}) = C' \text{ where } M \text{ transforms } C(k, M, \vec{w}) \text{ to } C'.$$

<u>Remark</u> This recursive definition requires that the length $\vec{w}$ is at least $n+1$ where n is the upper register index of M.

<u>Convention</u> If $\vec{w}$ is too short, we think of it as $\vec{w}$ where the rest of the entries is filled with $w_j = \varepsilon$.

<u>Remark</u> Computation sequence is always infinite.

<u>Def</u> The computation of M with input $\vec{w}$ <u>halts at time k (in k steps)</u> if k is the least number s.t.

$$C(k, M, \vec{w}) = (q_H, \vec{v})$$

for some $\vec{v}$.

If k does not exist, it doesn't halt.

If it halts, $\vec{v}$ is called the register content at time of halting.

We say M, M' are strongly equivalent if

① $\forall k \forall \vec{w}$ $C(k, M, \vec{w})$ and $C(k, M', \vec{w})$ have the same register content

② $\forall \vec{w}$ M halts after k steps with input $\vec{w}$

$\Longleftrightarrow$

M' halts after k steps with input $\vec{w}$

Observation If $|Q| = |Q'|$, then for every (Σ, Q, P), I find P' s.t. (Σ, Q, P) & (Σ, Q', P') are strongly eq.

Prop. 4.4 [Padding lemma]
For every M there are infinitely many diff. strongly eq. M'.

Pf. If $M = (\Sigma, Q, P)$ is a RM, it completely determines the computation seq. So if $\hat{q} \notin Q$, then $\hat{q}$ is never a state in the computation seq. So $\hat{Q} := Q \cup \{\hat{q}\}$ and $\hat{P} := P \cup \{\hat{q} \mapsto +(0, a, q_H)\}$ is strongly eq.

But $(\Sigma, \hat{Q}, \hat{P})$ has $|Q|+1$ states, so is different.

Iteration of this produces RM with

$$|Q|+n \text{ states}$$

for any $n \in \mathbb{N}$.

q.e.d.