# COMP30026 Models of Computation

Lecture 17: Introduction to Turing Machines

Mak Nazecic-Andrlon and William Umboh

Semester 2, 2024

Some material from Michael Sipser's slides

## Where are we?

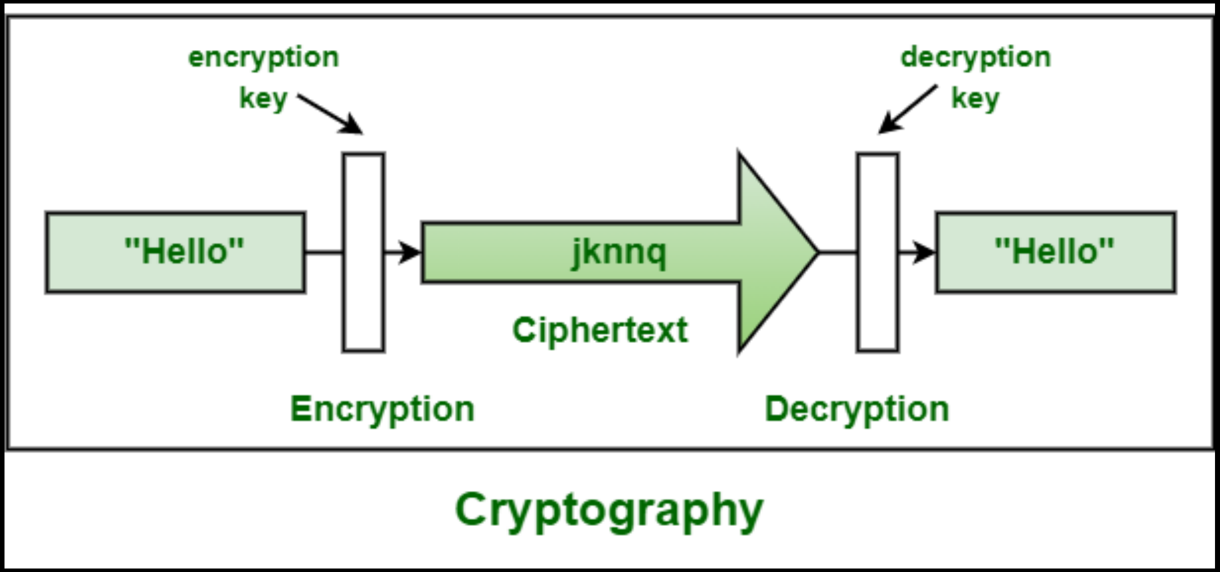
**Last few weeks:**
Restricted models of computation
- Regular languages: Finite automata, regular expressions
- Context-free languages: Pushdown automata, and context-free grammars

**Today:** (Sipser §3.1 - §3.2)
Turing machines (unrestricted model of computation)
- Turing-recognizable and Turing-decidable languages
- Church-Turing Thesis

Equivalence of variants of TMs
- Turing enumerators

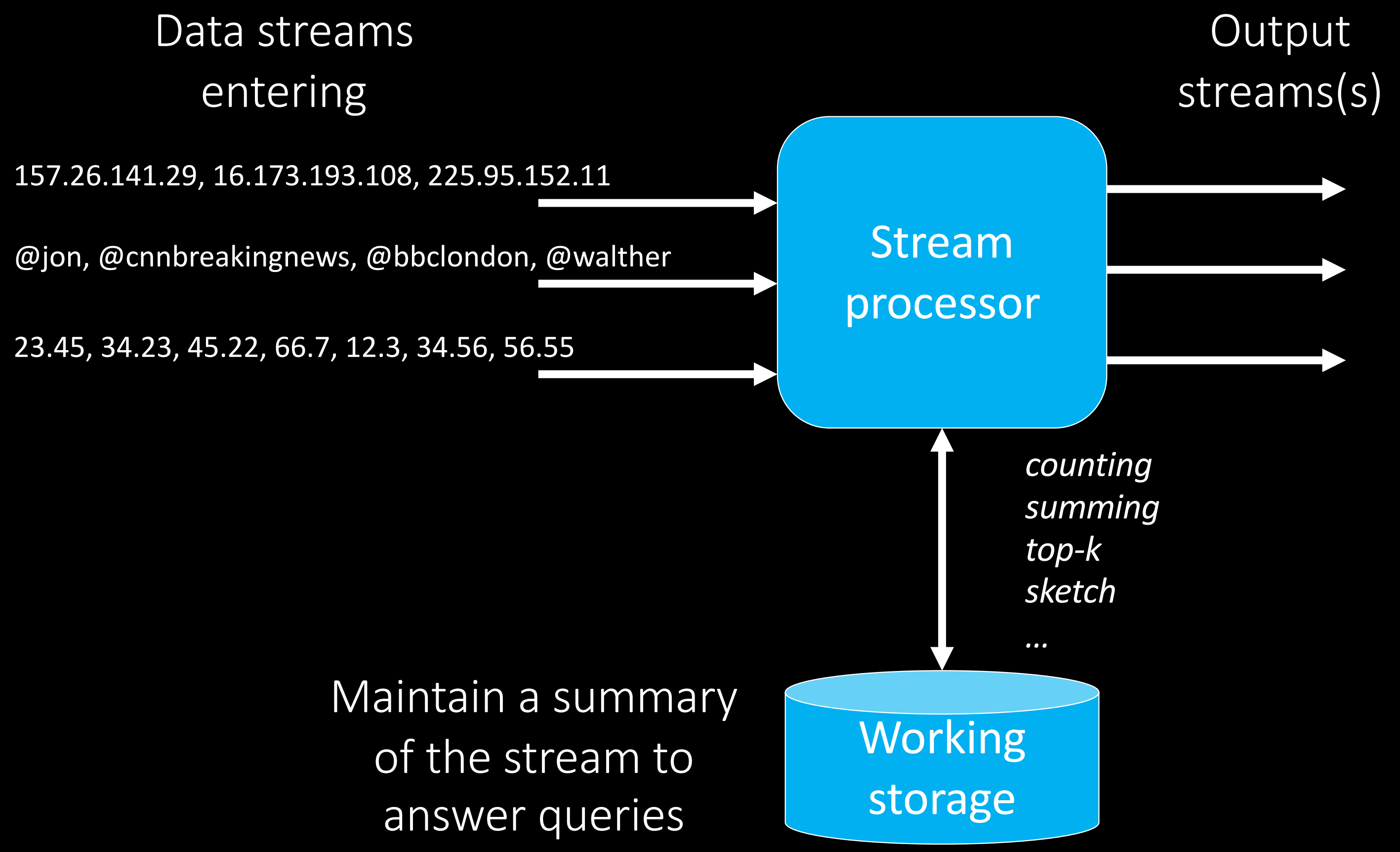
## Why study Turing machines?

1. Understand the limits of efficient computation



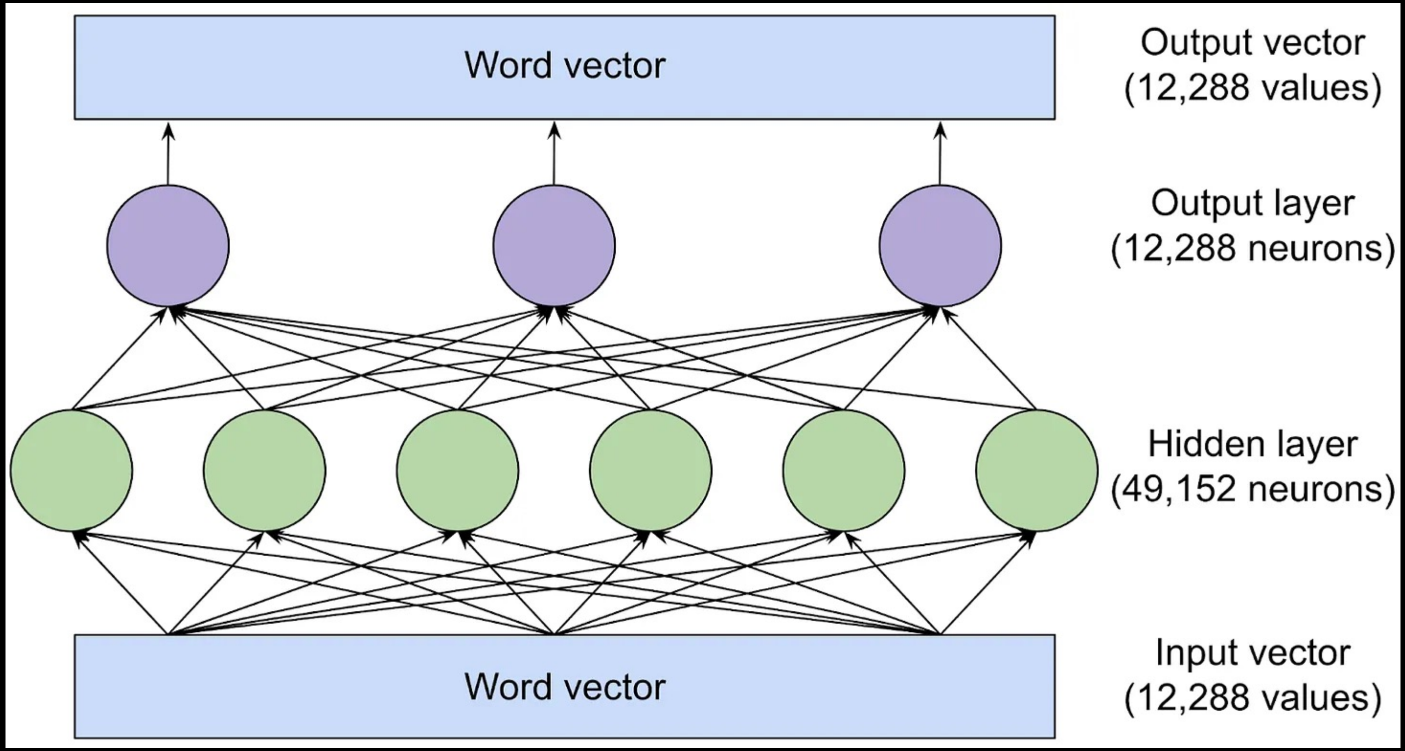
# Why study Turing machines?

2. Develop new models of computation to address challenges (streaming algorithms for Big Data)



## Why study Turing machines?

3. Understand the power of LLMs (yet another model of computation)


Word vector
Output vector (12,288 values)
Output layer (12,288 neurons)
Hidden layer (49,152 neurons)
Word vector
Input vector (12,288 values)

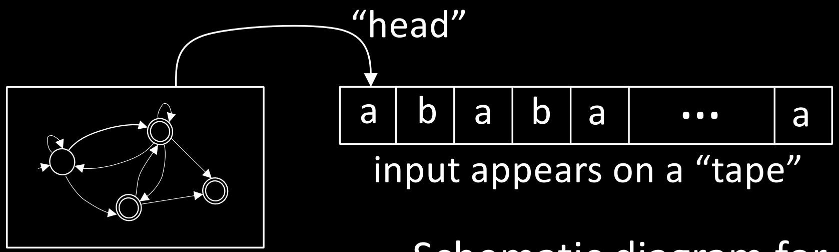
## Why study Turing machines?

For more on the impact of Theory of Computation, see Chapter 20 of Mathematics and Computation (Week 10 module on LMS) and other resources in my Ed post

# Previously, on Models of Computation

**Machine Model**

Finite Automata



Finite
control

Schematic diagram for DFA or NFA

Note: ”Memory” bounded by size of finite control

**Generative Model**

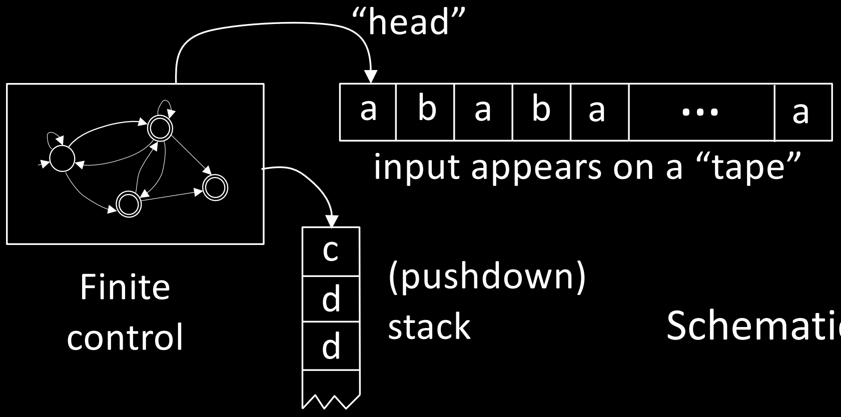
Regular Expression

$(0 \cup 1)^*$

# Previously, on Models of Computation

**Machine Model**

Pushdown Automata



Schematic diagram for PDA

**Generative Model**

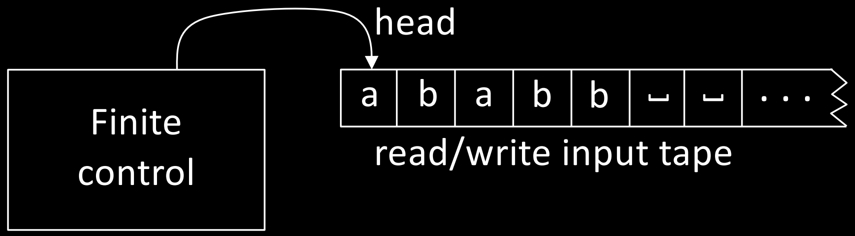
Context-Free Grammar

$$E \to E{+}T \mid T$$
$$T \to T{\times}F \mid F$$
$$F \to (\,E\,) \mid a$$

Operates like an NFA except can $\underbrace{\text{write-add}}_{\boxed{\text{push}}}$ or $\underbrace{\text{read-remove}}_{\boxed{\text{pop}}}$ symbols from the top of stack.

Note: "Memory" unbounded by size of finite control but still restricted (stack access)

# Turing Machines (TMs) - Informal



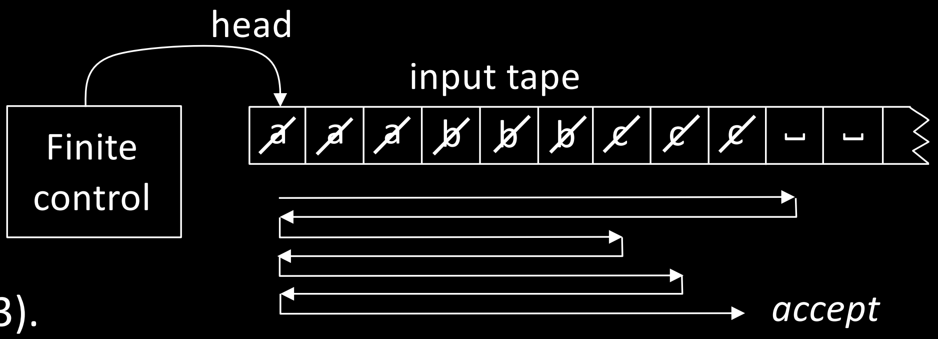
1) Head can read and write
2) Head is two way (can move left or right)
3) Tape is infinite (to the right)
4) Infinitely many blanks "␣" follow input
5) Can accept or reject any time (not only at end of input)

Tip for designing automata: pseudocode first, convert to formal automata spec later

# TM – Informal example

TM recognizing $B = \{\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k \mid k \geq 0\}$ (how to program this?)

1) Scan right until ␣ while checking if input is in $\mathrm{a}^*\mathrm{b}^*\mathrm{c}^*$, *reject* if not.
2) Return head to left end.
3) Scan right, crossing off single a, b, and c.
4) If the last one of each symbol, *accept*.
5) If the last one of some symbol but not others, *reject*.
6) If all symbols remain, return to left end and repeat from (3).



**Check-in 17.1**

How do we get the effect of “crossing off” with a Turing machine?

a) We add that feature to the model.

b) We use a tape alphabet $\Gamma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \cancel{\mathrm{a}}, \cancel{\mathrm{b}}, \cancel{\mathrm{c}}, ␣\}$.

c) All Turing machines come with an eraser.

## TM – Formal Definition

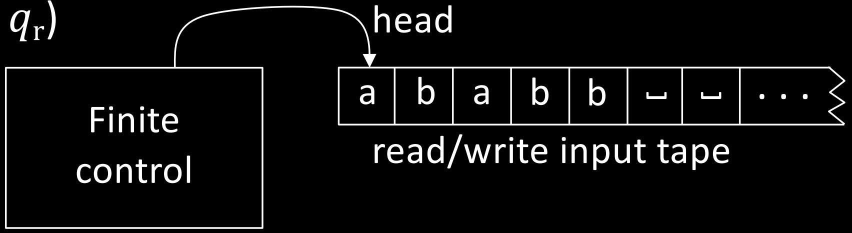
Defn: A Turing Machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

$\Sigma$ input alphabet

$\Gamma$ tape alphabet ($\Sigma \subseteq \Gamma$) incl. blank character ␣

$q_0$ initial state, $q_{acc}$ accept state, $q_{rej}$ reject state (sometimes $q_a$, $q_r$)

$\delta$: $Q\times\Gamma \rightarrow Q\times\Gamma\times \{L, R\}$ (L = Left, R = Right)



$$\delta(q, a) = (r, b, R)$$

If current state is $q$ and current symbol under tape head is a,

1. Change state to $r$
2. Over-write tape symbol a by b     ( b can be a )
3. Move the tape head to the right by one cell

# TM – Formal Definition

Defn: A Turing Machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$

$\Sigma$ input alphabet

$\Gamma$ tape alphabet $(\Sigma \subseteq \Gamma)$ incl. blank character $\sqcup$

$q_0$ initial state, $q_{\text{acc}}$ accept state, $q_{\text{rej}}$ reject state (sometimes $q_{\text{a}}$, $q_{\text{r}}$)

$\delta$: $\text{Q}\times\Gamma \to Q\times\Gamma\times$ {L, R} (L = Left, R = Right)

On input $w$ a TM $M$ may halt (enter $q_{\text{acc}}$ or $q_{\text{rej}}$) or $M$ may run forever ("loop").

So $M$ has 3 possible outcomes for each input $w$:

1. *Accept* $w$ (enter $q_{\text{acc}}$)
2. *Reject* $w$ by halting (enter $q_{\text{rej}}$)
3. *Reject* $w$ by looping (running forever)

## Check-in 17.2

This Turing machine model is deterministic.
How would we change it to be nondeterministic?

a) Add a second transition function.

b) Change $\delta$ to be $\delta$: $\text{Q}\times\Gamma \to \mathcal{P}(Q\times\Gamma\times$ {L, R} $)$

c) Change the tape alphabet $\Gamma$ to be infinite.

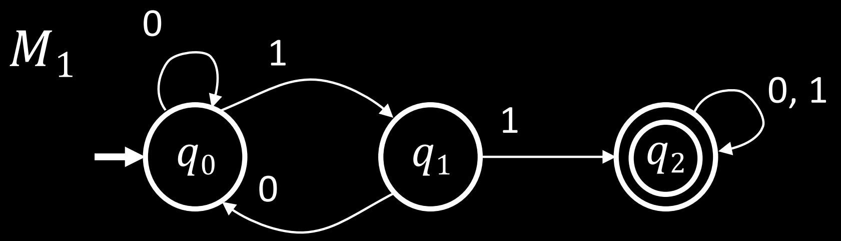
# Drawing TMs

We can have a graphical notation for Turing Machines similar to that for finite automata:

$\delta(q, \text{a}) = (r, \text{a}, \text{ R})$ equiv. $$\begin{tikzcd} q \arrow[r, "\text{a} \to \text{R}"] & r \end{tikzcd}$$

$\delta(q, \text{a}) = (r, \text{b}, \text{ R})$ equiv. $$\begin{tikzcd} q \arrow[r, "\text{a} \to \text{b, R}"] & r \end{tikzcd}$$

<u>Example</u> The following DFA recognizes language $\{w \mid w \text{ contains substring } 11\}$



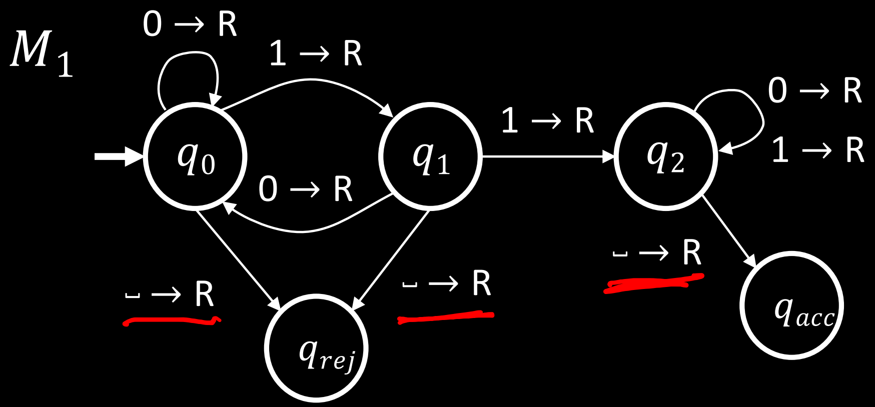
# Drawing TMs

We can have a graphical notation for Turing Machines similar to that for finite automata:

$\delta(q, \mathrm{a}) = (r, \mathrm{a},\ \mathrm{R})$ equiv. $$\begin{tikzcd} q \arrow[r, "\mathrm{a} \rightarrow \mathrm{R}"] & r \end{tikzcd}$$

$\delta(q, \mathrm{a}) = (r, \mathrm{b},\ \mathrm{R})$ equiv. $$\begin{tikzcd} q \arrow[r, "\mathrm{a} \rightarrow \mathrm{b},\ \mathrm{R}"] & r \end{tikzcd}$$

Example The following TM recognizes language $\{w \mid w \text{ contains substring } 11\}$

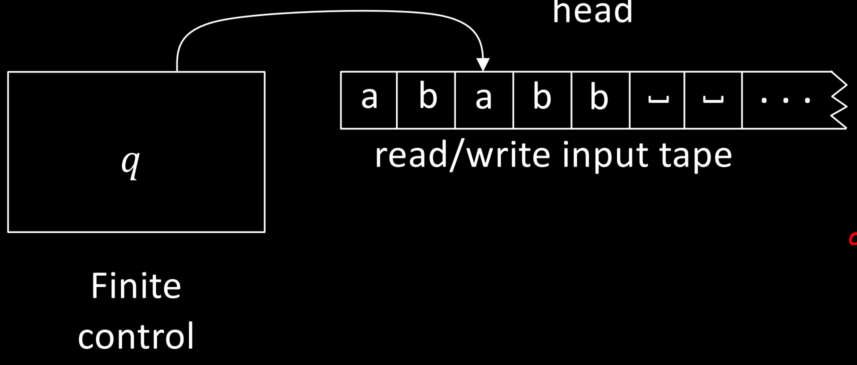


## Turing Machine Configurations

The configuration of a TM is a "snapshot of its execution at a point in time":

- Current state
- Current state of the tape
- Current location of the tape head

Example



abqabb

This configuration is represented using the notation ab$q$abb

# Turing Machine Computation Formally

Notation: If applying $\delta$ to config $C$ yields config $C'$ write $C \Rightarrow C'$

Examples

$\text{u}q\text{bv} \Rightarrow \text{uc}t\text{v}$ if $\delta(q, \text{b}) = (t, \text{c}, R)$

$\text{u}q\text{bv} \Rightarrow t\text{ucv}$ if $\delta(q, \text{b}) = (t, \text{c}, L)$

$q\text{buv} \Rightarrow t\text{cuv}$ if $\delta(q, \text{b}) = (t, \text{c}, L)$ (tape head can't move to left if it's already at the start)

Start configuration of $M$ on input $w$ is $q_0 w$

Defn. $M$ accepts $w$ iff there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that

1. $C_1 = q_0 w$
2. $C_i \Rightarrow C_{i+1}$ for every $i$ from 1 to $k$
3. State of $C_k$ is $q_{acc}$

# TM Recognizers and Deciders

Let $M$ be a TM. Then $L(M) = \{w|\ M \text{ accepts } w\}$.

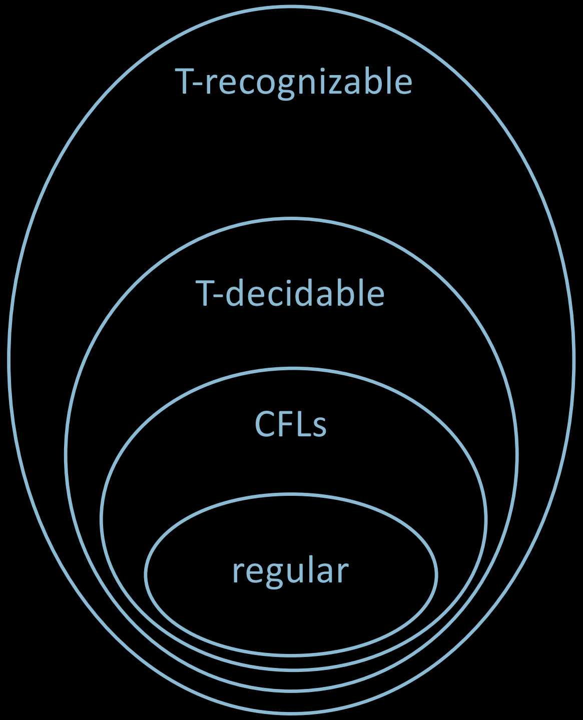
Say that $M$ recognizes $A$ if $A = L(M)$.

**Defn:** $A$ is Turing-recognizable if $A = L(M)$ for some TM $M$ (aka recursively enumerable).

**Defn:** TM $M$ is a decider if $M$ halts on all inputs.

Say that $M$ decides $A$ if $A = L(M)$ and $M$ is a decider.

**Defn:** $A$ is Turing-decidable if $A = L(M)$ for some TM decider $M$.



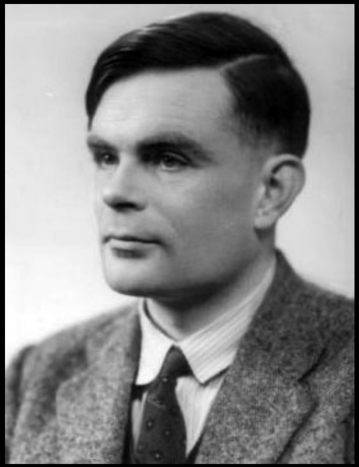
# Church-Turing Thesis ~1936

Alonzo Church
1903–1995

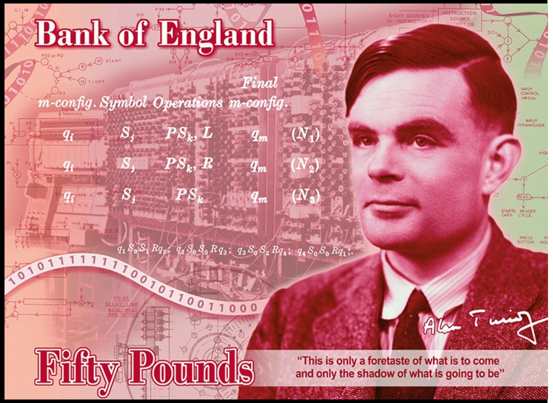
| Algorithm | = | Turing machine |
|---|---|---|
| Intuitive | | Formal |

Instead of Turing machines,
can use any other “reasonable” model
of unrestricted computation:
$\lambda$-calculus, random access machine,
your favorite programming language, …

Big impact on mathematics.


Alan Turing
1912–1954

Will appear in 2021


Bank of England
Final
m-config. Symbol Operations m-config.
Fifty Pounds
"This is only a foretaste of what is to come and only the shadow of what is going to be"

## Hilbert’s 10th Problem

**In 1900 David Hilbert posed 23 problems**

#2) Prove that the axioms of mathematics are consistent.

#10) Give an algorithm for solving *Diophantine equations.*

**Diophantine equations:**

Equations of polynomials where solutions must be integers.

Example: $3x^2 - 2xy - y^2z = 7$ integer solution: $x = 1,\ y = 2,\ z = -2$

Let $D = \{p|$ polynomial $p(x_1, x_2, \ldots, x_k) = 0$ has a solution in integers)

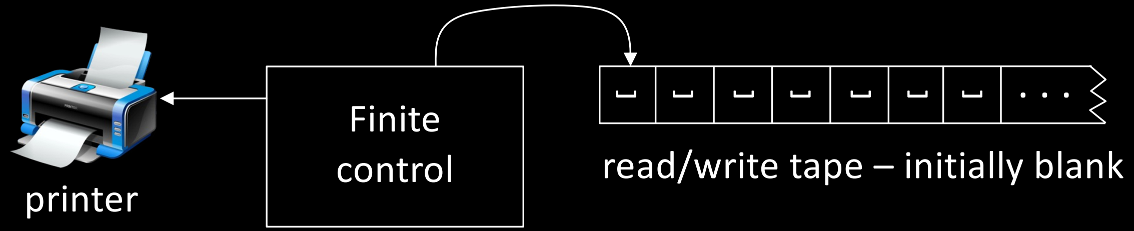
Hilbert’s 10th problem: Give an algorithm to decide $D$.

Matiyasevich proved in 1970: $D$ is not decidable.

Exercise: $D$ is T-recognizable.

David Hilbert
1862—1943

# Turing Enumerators



**Defn:** A <u>Turing Enumerator</u> is a deterministic TM with a printer.
It starts on a blank tape and it can print strings $w_1, w_2, w_3, \ldots$ possibly going forever.
Its language is the set of all strings it prints. It is a generator, not a recognizer.
For enumerator $E$ we say $L(E) = \{w \mid E \text{ prints } w\}$.

**Theorem:** A is Turing-recognizable iff $A = L(E)$ for some Turing-enumerator $E$.

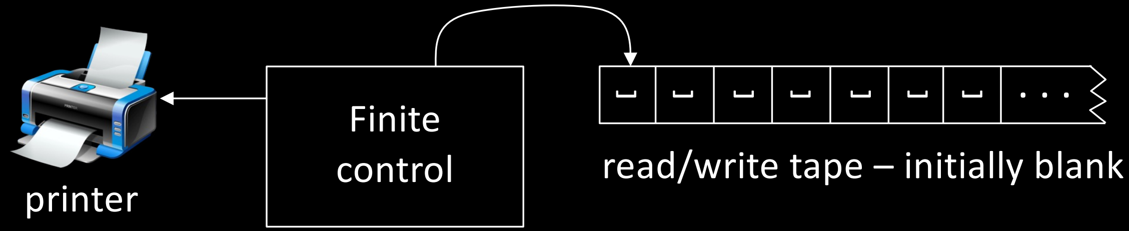
**Proof:** (←) Convert $E$ to equivalent TM $M$.

```
M = for input w:
        Simulate E (on blank input).
        Whenever E prints x, test x = w.
        Accept if = and continue otherwise.
```

**Proof:** (→) Convert TM $M$ to equivalent enumerator $E$.

```
E = Simulate M on each w_i in Σ* = {ε, 0,1,00,01,10, … }
        If M accepts w_i then print w_i .
        Continue with next w_i .
    Problem: What if M on w_i loops?
    Fix: Simulate M on w_1, w_2, …, w_i for i steps, for i = 1,2, …
         Print those w_i which are accepted.
```

# Turing Enumerators



**Defn:** A Turing Enumerator is a deterministic TM with a printer.
It starts on a blank tape and it can print strings $w_1 , w_2 , w_3 , \ldots$ possibly going forever.
Its language is the set of all strings it prints. It is a generator, not a recognizer.
For enumerator $E$ we say $L(E) = \{w| \; E \text{ prints } w\}$.

**Theorem:** A is Turing-recognizable iff $A = L(E)$ for some Turing-enumerator $E$.

**Check-in 17.3**
When converting TM $M$ to enumerator $E$,
does $E$ always print the strings in ***string order***?
a) Yes.
b) No.

**Proof:** (→) Convert TM $M$ to equivalent enumerator $E$.
$E =$ Simulate $M$ on each $w_i$ in $\Sigma^* = \{\varepsilon, 0,1,00,01,10, \ldots\}$
If $M$ accepts $w_i$ then print $w_i$ .
Continue with next $w_i$ .
*Problem:* What if $M$ on $w_i$ loops?
*Fix:* Simulate $M$ on $w_1 , w_2 , \ldots , w_i$ for $i$ steps, for $i = 1,2, \ldots$
Print those $w_i$ which are accepted.

## Quick review of today

1. Defined Turing machines (TMs).
2. Defined TM deciders (halt on all inputs).
3. T-recognizable and T-decidable languages.
4. Church-Turing Thesis
5. Equivalence of variants of TMs (Enumerators)