Context-sensitive and decidable languages

Sides based on material by Kani, Erickson, Chekuri, et. al.

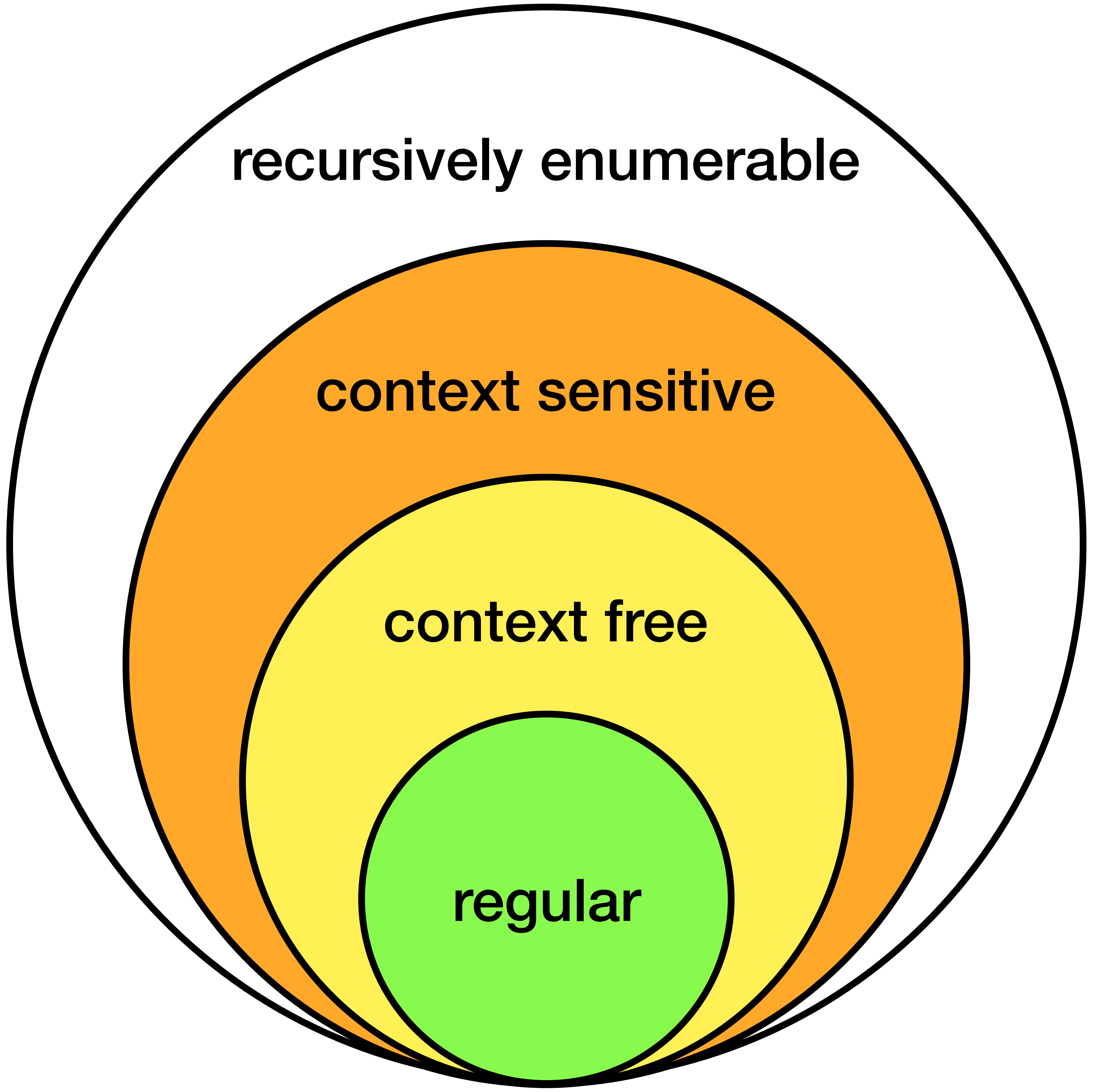
All mistakes are my own! - Ivan Abraham (Fall 2024)

Image by ChatGPT (probably collaborated with DALL-E)

Context Sensitive Language

Definition

A language L is said to be context-sensitive if there exists a context-sensitive grammar G, such that $L = L(G)$.



Context Sensitive Grammar (CSG)

Definition

A ~~CFG~~ is a quadruple $G = (V, T, P, S)$

$V \rightarrow$ finite set of non-terminals

$T \rightarrow$ " " " terminals

$P \rightarrow \quad P \subseteq V \times (V \cup T)^*$

rules have the form

$V \rightarrow \alpha \quad , \quad \alpha \in (V \cup T)^*$

$S \rightarrow$ start symbol.

Context Sensitive Grammar (CSG)

Definition

A CFG is a quadruple $G = (V, T, P, S)$

- V is a finite set of *non-terminal symbols.*
- T is a finite set of *terminal symbols* (alphabet).
- P is a finite set of *productions*, each of the form $A \to \alpha$ where $A \in V$ and α is a string in $(V \cup T)^*$.
- $S \in V$ is a *start symbol.*

Context Sensitive Grammar (CSG)

Definition

A CSG is a quadruple $G = (V, T, P, S)$

- V is a finite set of *non-terminal symbols.*
- T is a finite set of *terminal symbols* (alphabet).
- P is a finite set of *productions*, each of the form $\alpha \rightarrow \beta$ where α and β are strings in $(V \cup T)^*$.
- $S \in V$ is a *start symbol.*

Context Sensitive Grammar (CSG)

Example formally for completeness

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

Context Sensitive Grammar (CSG)

Example formally for completeness

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

$$V = \{S, A, B\}$$

Context Sensitive Grammar (CSG)

Example formally for completeness

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

$$V = \{S, A, B\}$$

$$T = \{a, b, c\}$$

$S \to abc \mid aAbC$

$Ab \to bA$

$Ac \to Bbcc$

$bB \to Bb$

$aB \to aa \mid$ ____

abAc.

abBbcc

aBbbcc

aabbcc

Context Sensitive Grammar (CSG)

Example formally for completeness

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

$$V = \{S, A, B\}$$

$$T = \{a, b, c\}$$

$$P = \left\{ \begin{array}{r} S \to abc \mid aAbc, \\ Ab \to bA, \\ Ac \to Bbcc, \\ bB \to Bb, \\ aB \to aa \mid aaA \end{array} \right\}$$

Context Sensitive Grammar (CSG)

Example formally for completeness

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

$$V = \{S, A, B\}$$

$$T = \{a, b, c\}$$

$$P = \left\{ \begin{aligned} S &\to abc \mid aAbc, \\ Ab &\to bA, \\ Ac &\to Bbcc, \\ bB &\to Bb, \\ aB &\to aa \mid aaA \end{aligned} \right\}$$

$$G = \left(\{S, A, B\}, \{a, b, c\}, \left\{ \begin{aligned} S &\to abc \mid aAbc, \\ Ab &\to bA, \\ Ac &\to Bbcc, \\ bB &\to Bb, \\ aB &\to aa \mid aaA \end{aligned} \right\}, S \right)$$

$L = \{a^n b^n c^n \mid n \geq 1\}$

Context Sensitive Language

Example formally for completeness

$$V = \{S, A, B\}$$

$$T = \{a, b, c\}$$

$$P = \left\{ \begin{array}{r} S \rightarrow abc \mid aAbc, \\ Ab \rightarrow bA, \\ Ac \rightarrow Bbcc, \\ bB \rightarrow Bb, \\ aB \rightarrow aa \mid aaA \end{array} \right\}$$

$L = \{a^n b^n c^n \mid n \geq 1\}$

Context Sensitive Language

Example formally for completeness

$V = \{S, A, B\}$

$T = \{a, b, c\}$

$$P = \left\{ \begin{aligned} S &\rightarrow abc \mid aAbc, \\ Ab &\rightarrow bA, \\ Ac &\rightarrow Bbcc, \\ bB &\rightarrow Bb, \\ aB &\rightarrow aa \mid aaA \end{aligned} \right\}$$

$S \rightsquigarrow aAbc \rightsquigarrow abAc \rightsquigarrow abBbcc \rightsquigarrow aBbbcc$

$\rightsquigarrow aaAbbcc \rightsquigarrow aabAbcc \rightsquigarrow aabbAcc$

$\rightsquigarrow aabbBbccc \rightsquigarrow aabBbbccc$

$\rightsquigarrow aaBbbbccc \rightsquigarrow aaabbbccc$

$$L = \{a^n b^n c^n \mid n \geq 1\}$$

Context Sensitive Language

Another solution

$S \to aTU \mid aSTU$

$aT \to ab$

$bU \to bc$

$UT \to TU$

$bT \to bb$

$bU \to bc$

$cU \to cc$

Apologies for the "colour coding", Hard to do it on the fly but leaving it unchanged so it matches the recording

$aTU \to abU \to abc$

$aSTU \to aaTUTU$

$\downarrow$

$aabUTU$

$\downarrow$

$aabTUU$

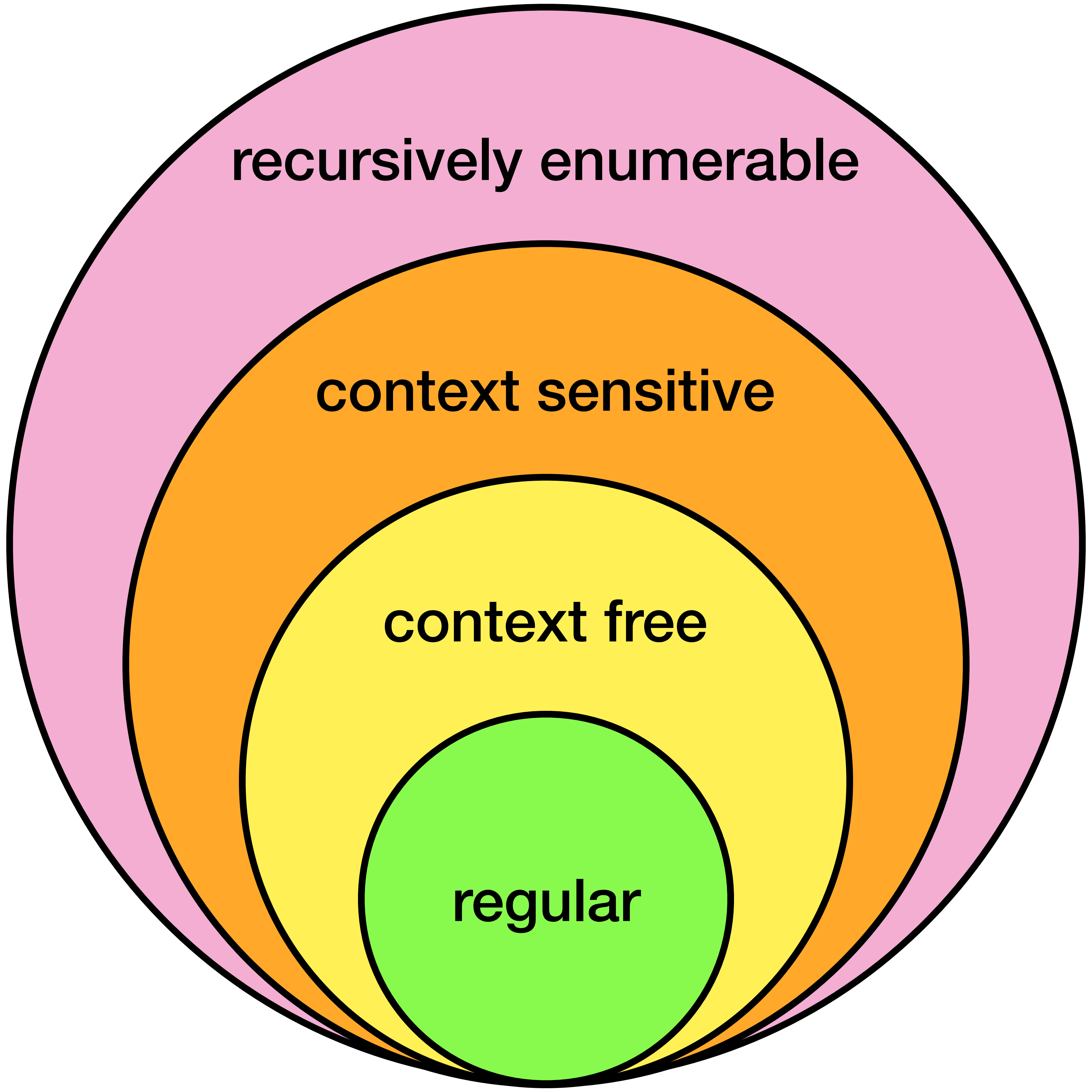
$aabbUU$

$aabbcU$

$aabbcc$

Recursively enumerable (Turing Recognizable) language

Recursively enumerable (Turing Recognizable) language



“Most general” computer

- Is there a kind of computer that can accept any language or compute any function?

“Most general” computer

- Is there a kind of computer that can accept any language or compute any function?

- Recall counting argument. Set of all languages: $\{L \,|\, L \subseteq \{0,1\}^*\}$ is

"Most general" computer

- Is there a kind of computer that can accept any language or compute any function?

- Recall counting argument. Set of all languages: $\{L \mid L \subseteq \{0,1\}^*\}$ is

(a) countably infinite (b) uncountably infinite

“Most general” computer

- Is there a kind of computer that can accept any language or compute any function?

- Recall counting argument. Set of all languages: $\{L \mid L \subseteq \{0,1\}^*\}$ is

(a) countably infinite (b) uncountably infinite

- Set of all programs: $\{P \mid P \text{ is a finite length computer program}\}$ is

"Most general" computer

- Is there a kind of computer that can accept any language or compute any function?

- Recall counting argument. Set of all languages: $\{L \mid L \subseteq \{0,1\}^*\}$ is

  (a) countably infinite (b) uncountably infinite

- Set of all programs: $\{P \mid P \text{ is a finite length computer program}\}$ is

  (a) countably infinite (b) uncountably infinite

There is a gap b/w # of computer programs and all problems.

“Most general” computer

- Is there a kind of computer that can accept any language or compute any function?

- Recall counting argument. Set of all languages: $\{L \mid L \subseteq \{0,1\}^*\}$ is

(a) countably infinite (b) uncountably infinite

- Set of all programs: $\{P \mid P \text{ is a finite length computer program}\}$ is

(a) countably infinite (b) uncountably infinite

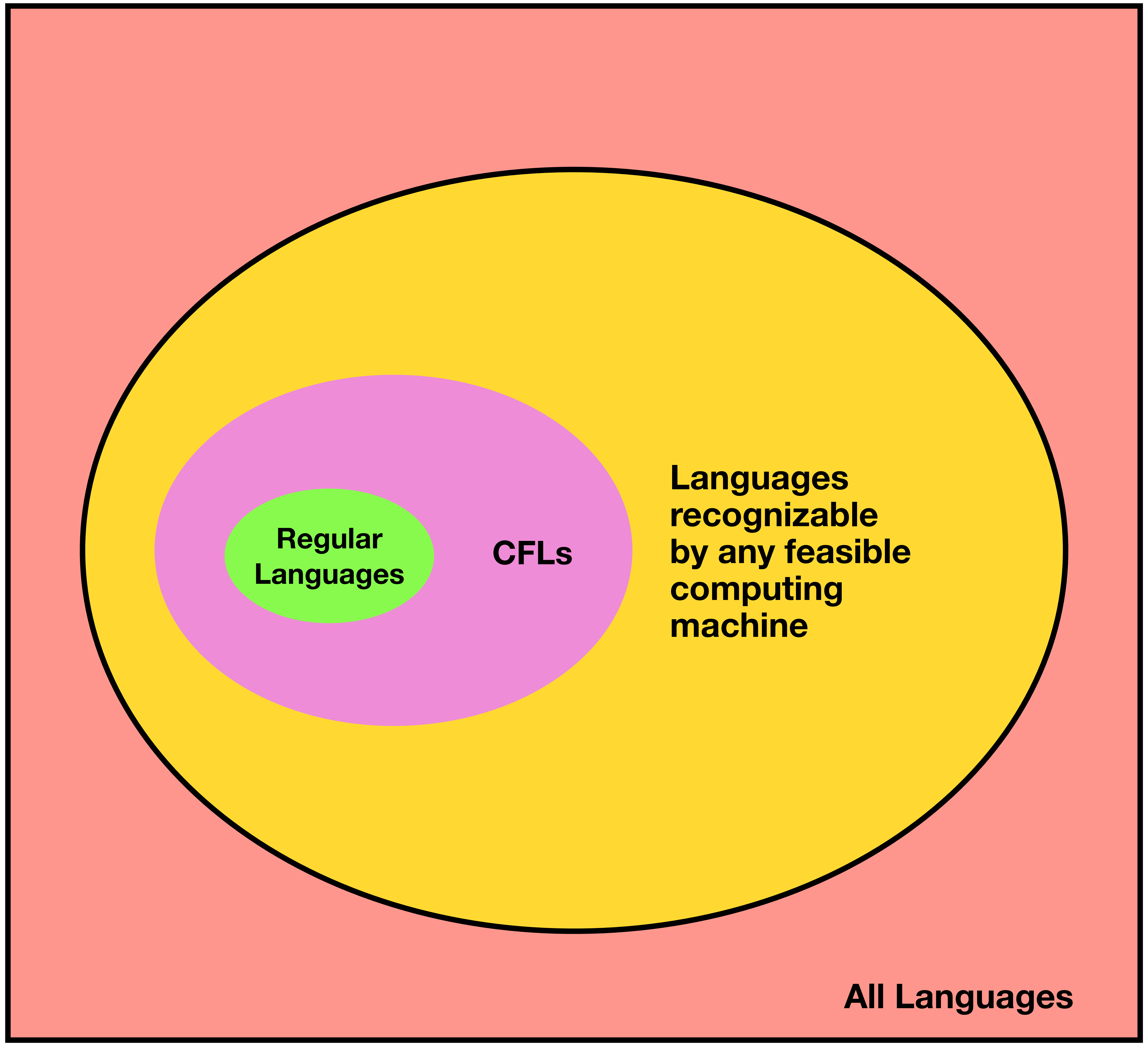
- **Conclusion: There are languages for which there are no programs.**

“Most general” computer

- We may need unbounded memory to recognize context-free languages.

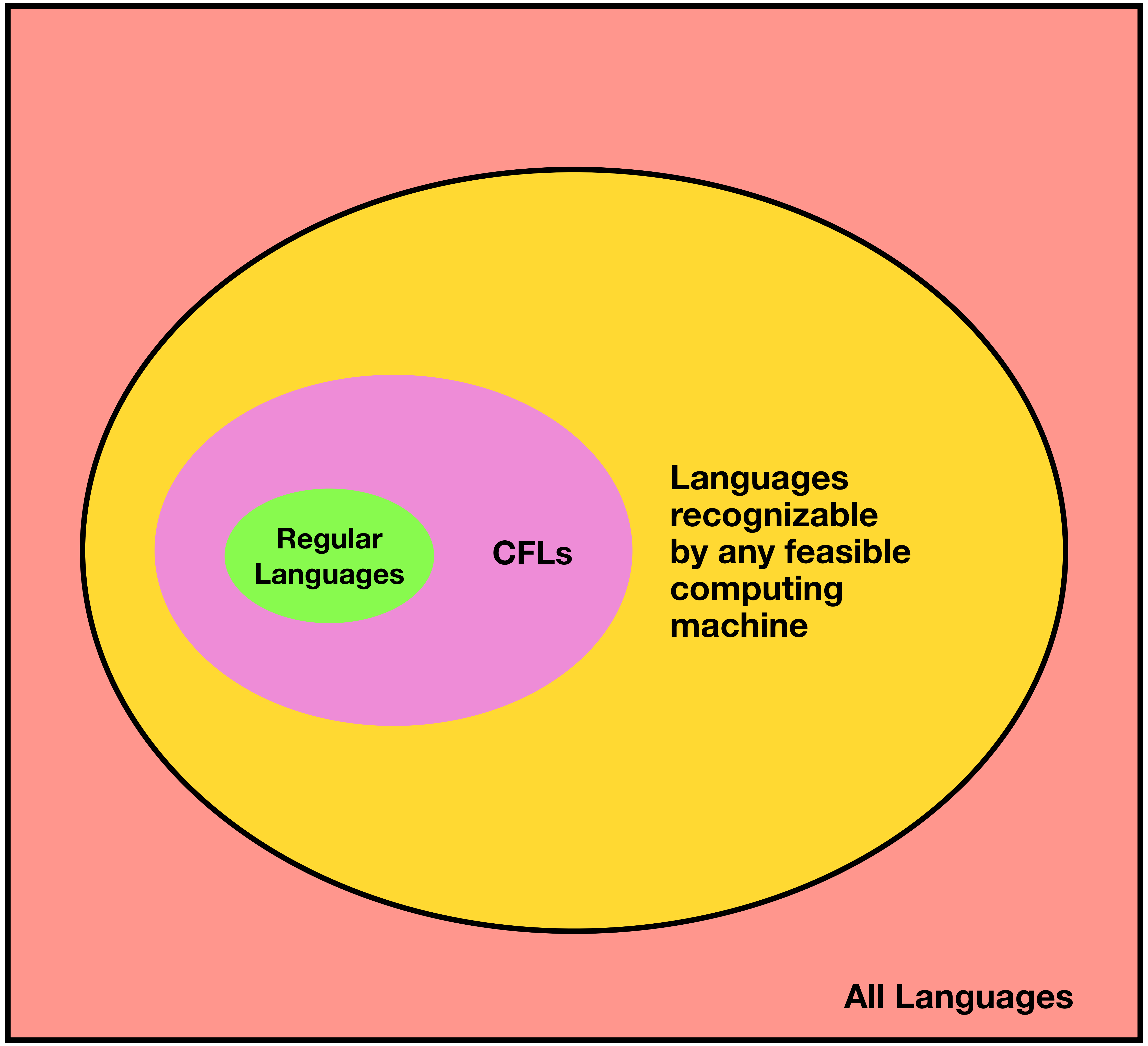
“Most general” computer

- We may need unbounded memory to recognize context-free languages.



"Most general" computer

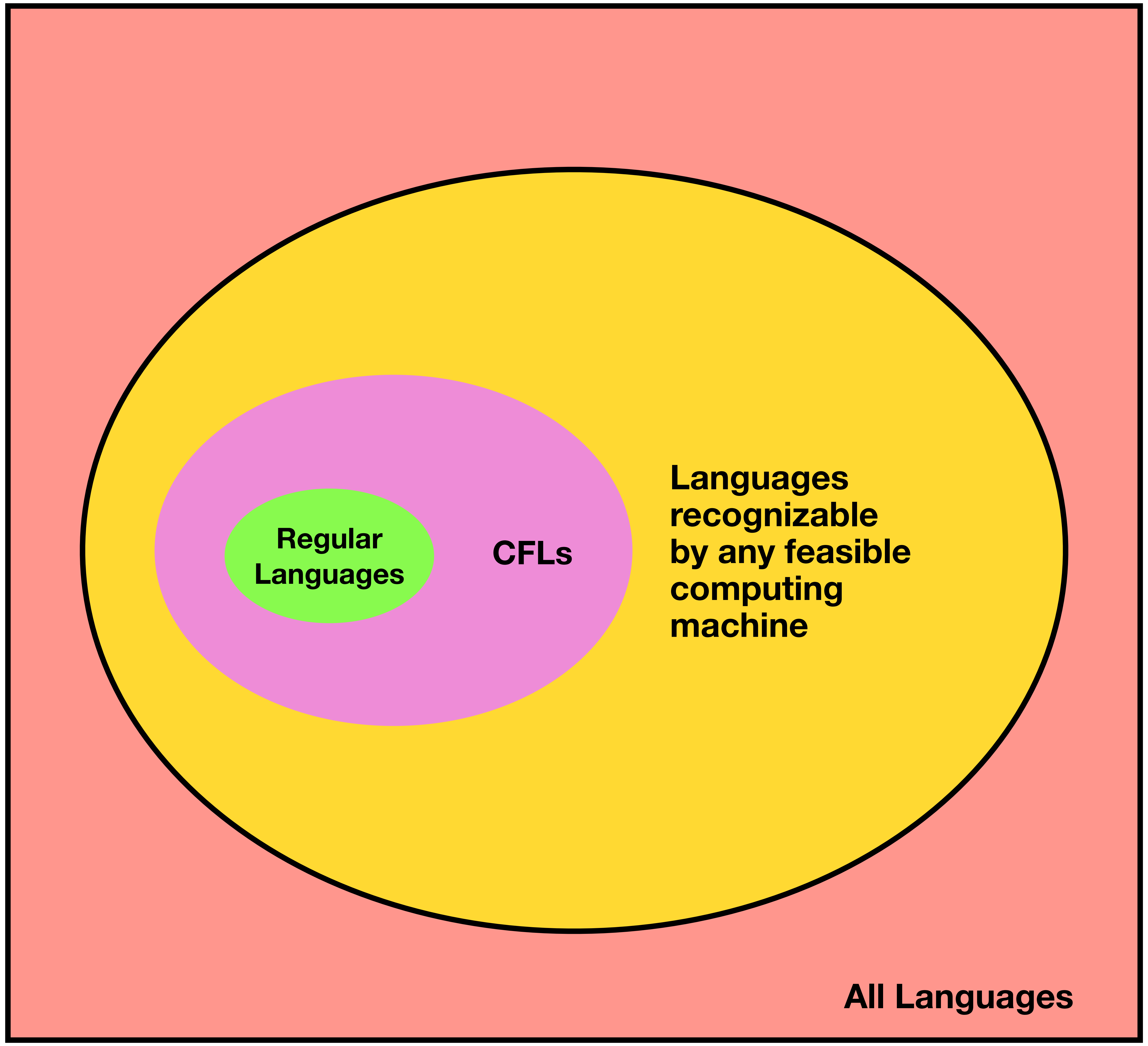
- We may need unbounded memory to recognize context-free languages.
    - E.g. $\{a^n b^n \mid n \in \mathbb{N}\}$ requires unbounded counting.



“Most general” computer

- We may need unbounded memory to recognize context-free languages.
  - E.g. $\{a^n b^n \mid n \in \mathbb{N}\}$ requires unbounded counting.

How do we model a computing device that has unbounded memory?

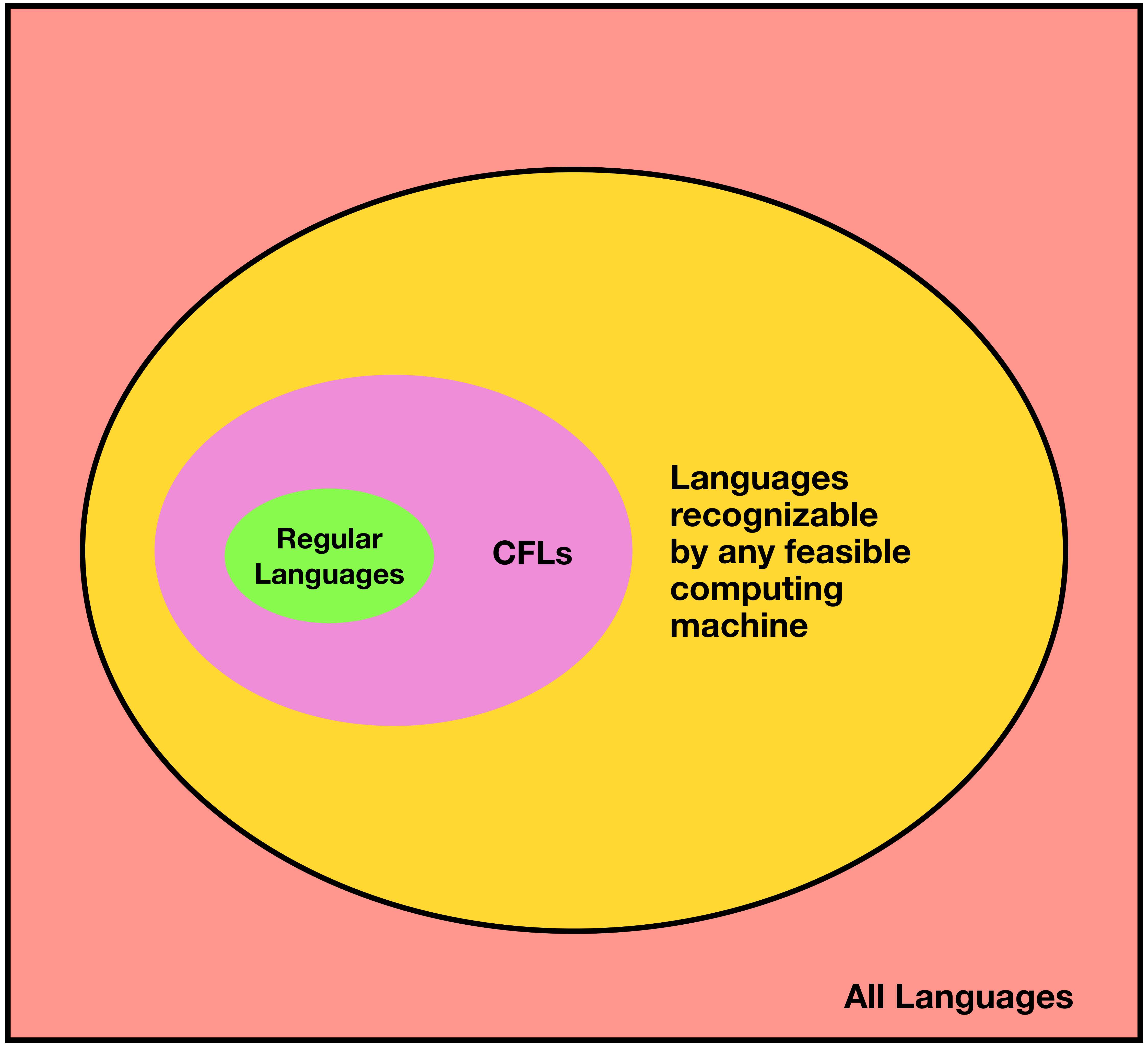


“Most general” computer

- We may need unbounded memory to recognize context-free languages.
  - E.g. $\{a^n b^n \mid n \in \mathbb{N}\}$ requires unbounded counting.

How do we model a computing device that has unbounded memory?

OR



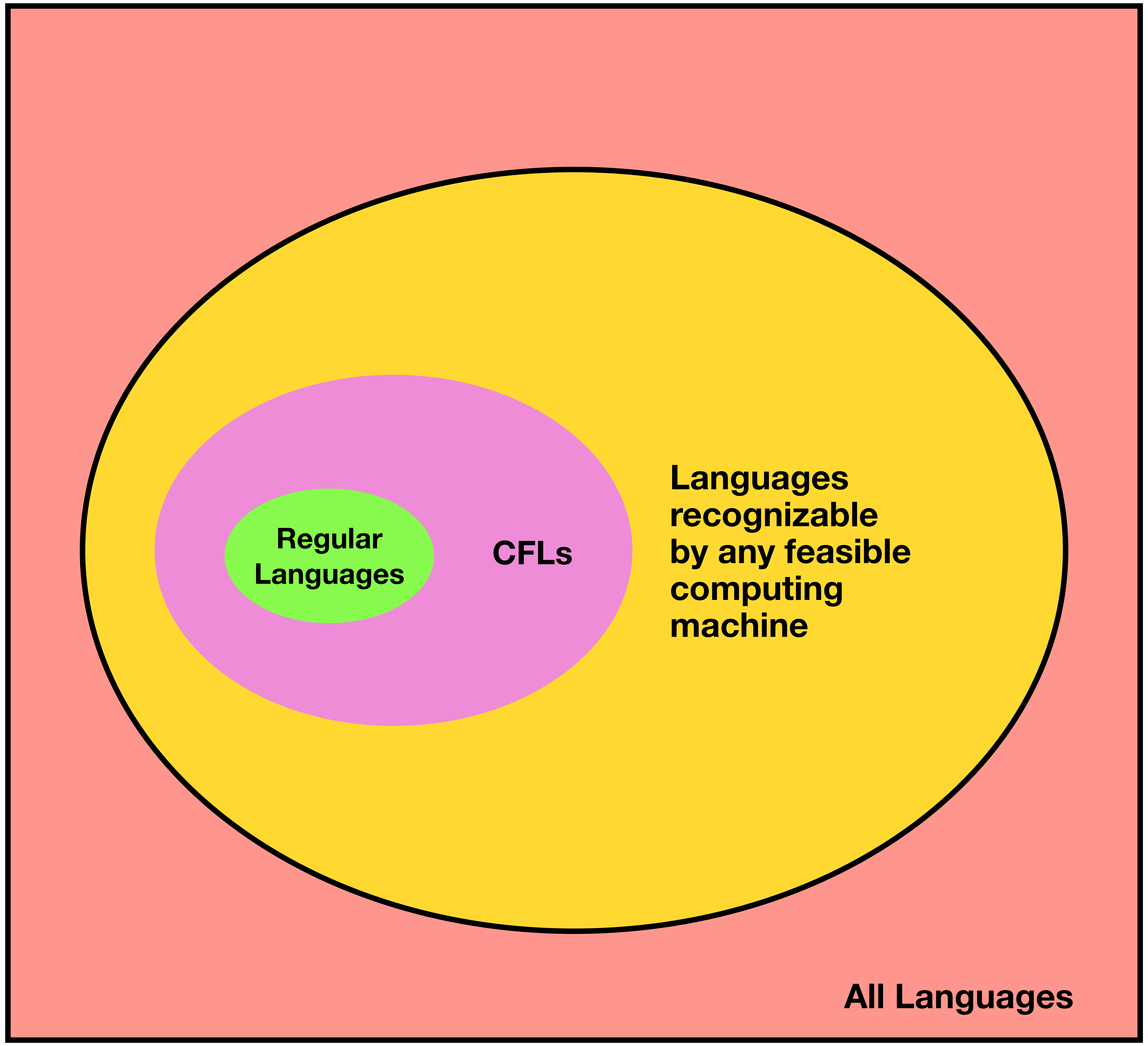
"Most general" computer

- We may need unbounded memory to recognize context-free languages.
  - E.g. $\{a^n b^n \mid n \in \mathbb{N}\}$ requires unbounded counting.

How do we model a computing device that has unbounded memory?

OR

How do we model a computing device that can recognize as many languages as possible?

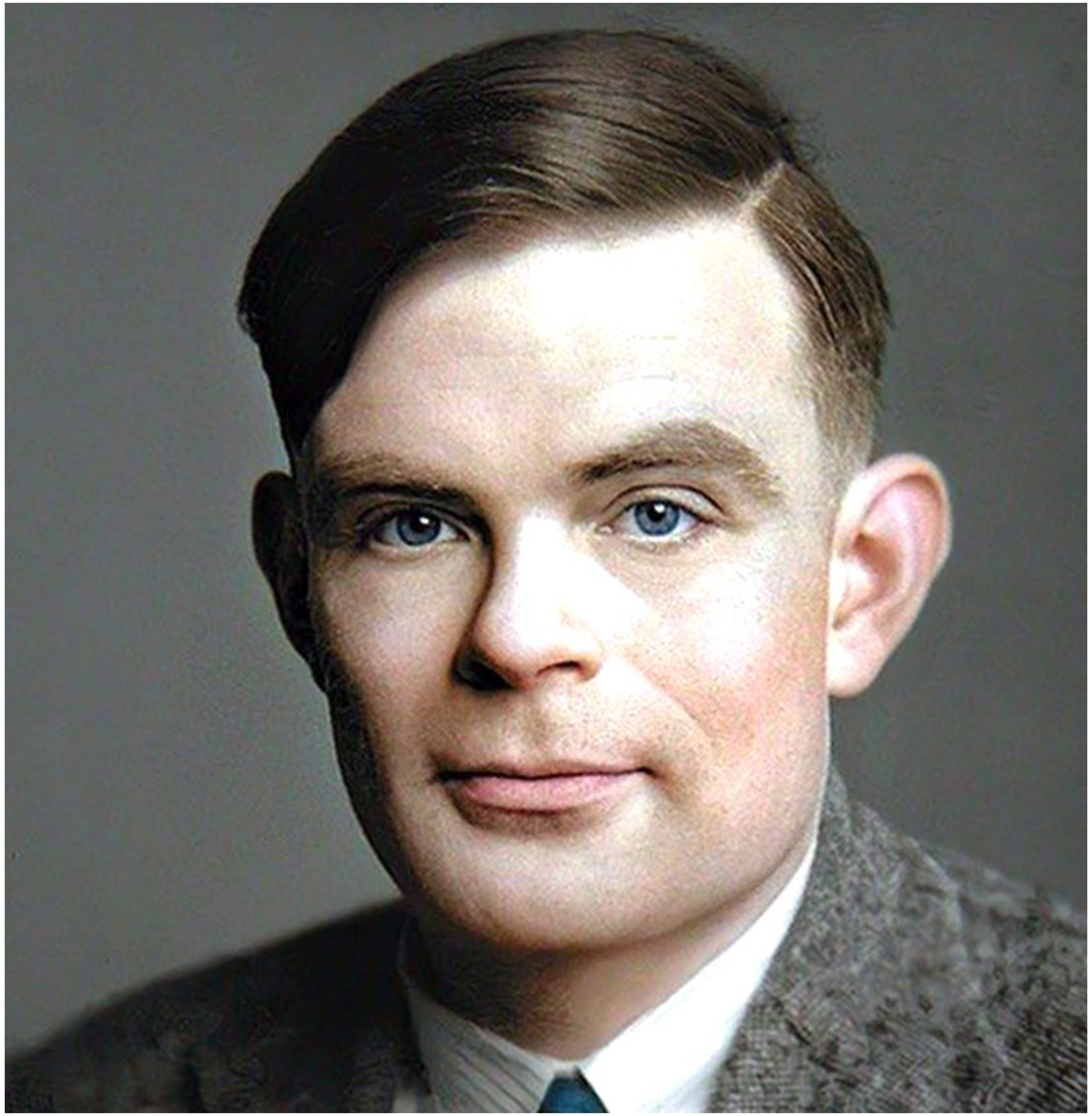


Turing Machine

A brief history

Turing Machine
A brief history

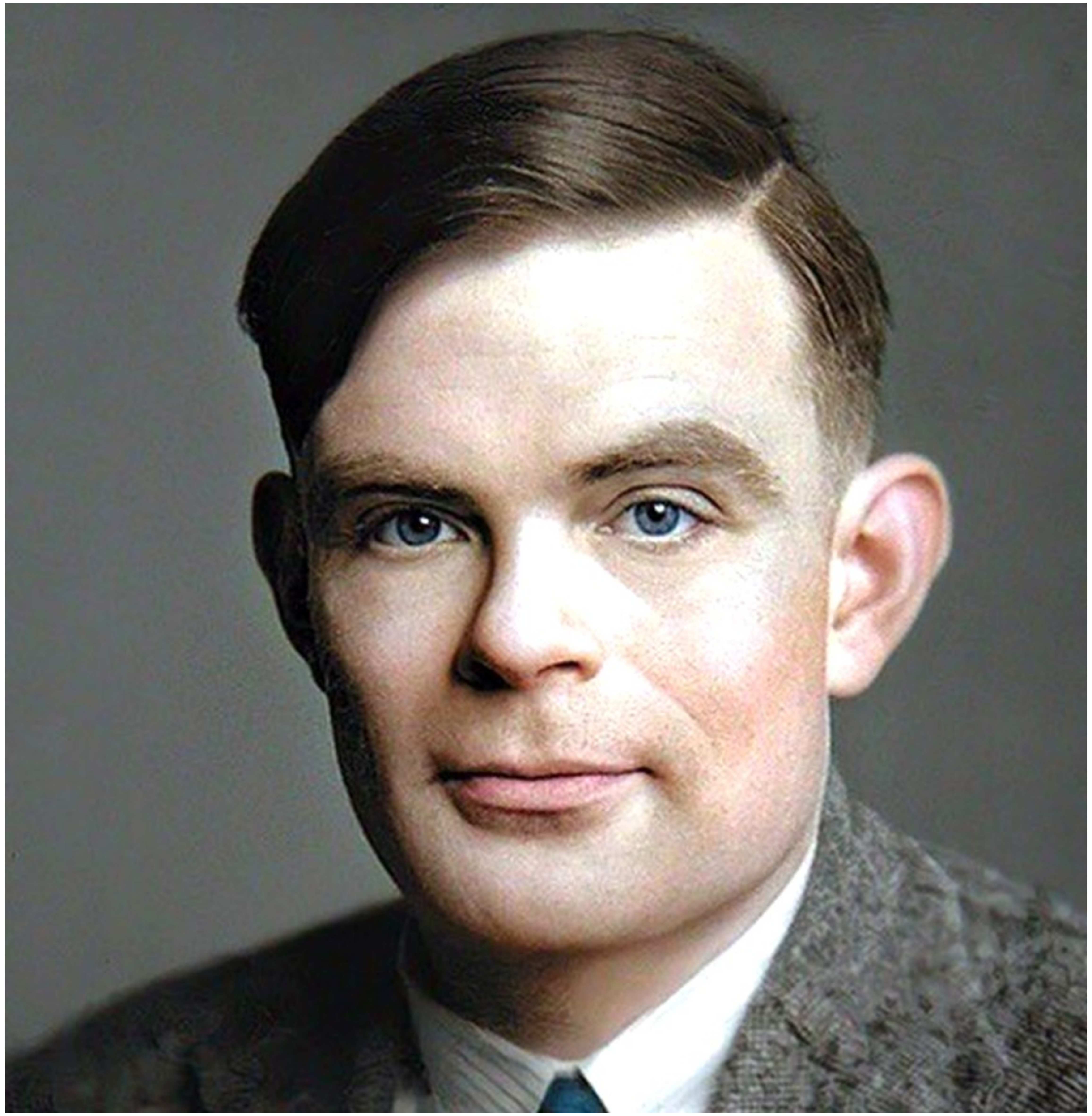
- In March 1936, Alan Turing (aged 23!) published a paper detailing the **a-machine** (for **automatic machine**), an automaton for computing on real numbers.



Turing Machine

A brief history

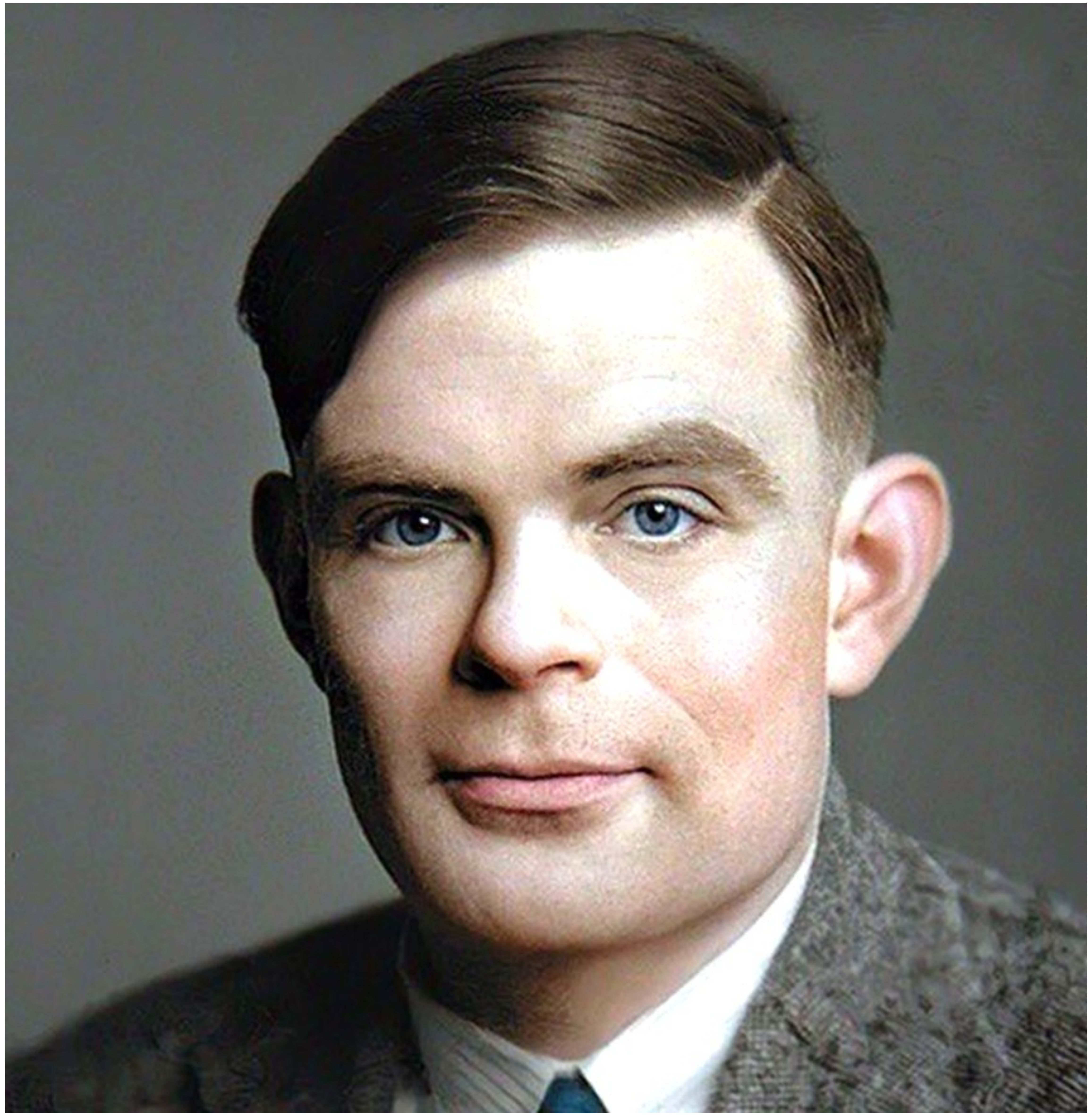
- In March 1936, Alan Turing (aged 23!) published a paper detailing the **a-machine** (for **automatic machine**), an automaton for computing on real numbers.
- They're now more popularly referred to as **Turing machines** in his honor.



Turing Machine

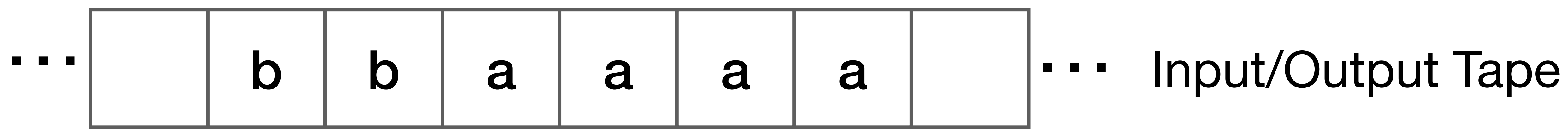
A brief history

- In March 1936, Alan Turing (aged 23!) published a paper detailing the **a-machine** (for **automatic machine**), an automaton for computing on real numbers.
- They're now more popularly referred to as **Turing machines** in his honor.
- Watch: The Imitation Game!



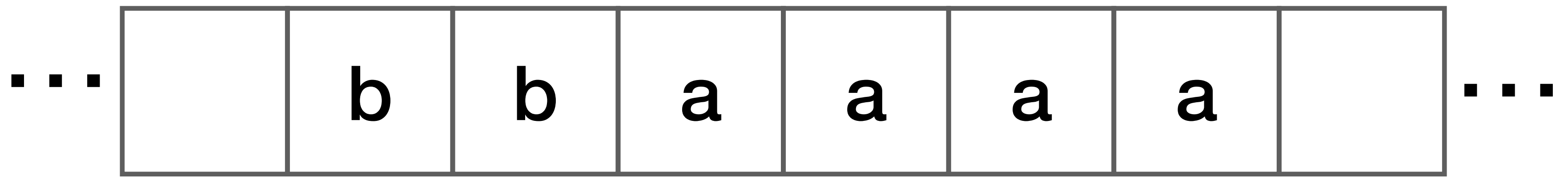
Turing Machine

Turing Machine



- Input written on (infinite) one sided tape.

Turing Machine

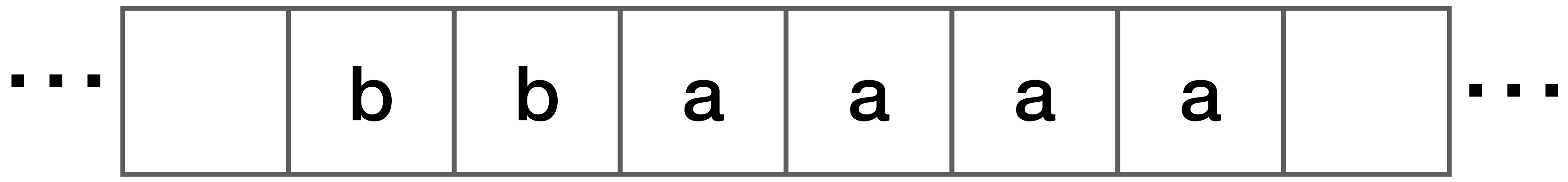


Input/Output Tape

allow blanks ("erase") on memory/tape.

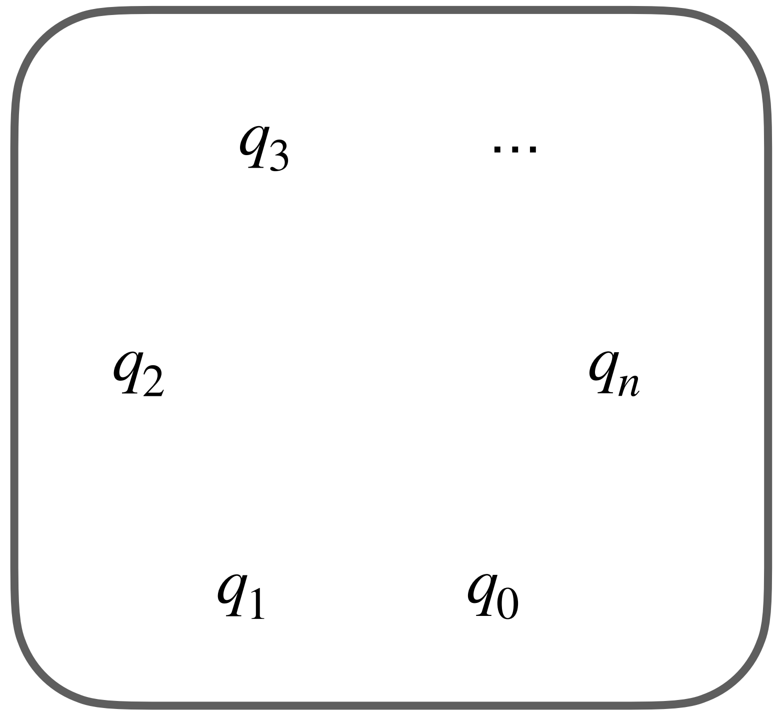
- Input written on (infinite) one sided tape.
- Special blank characters.

Turing Machine



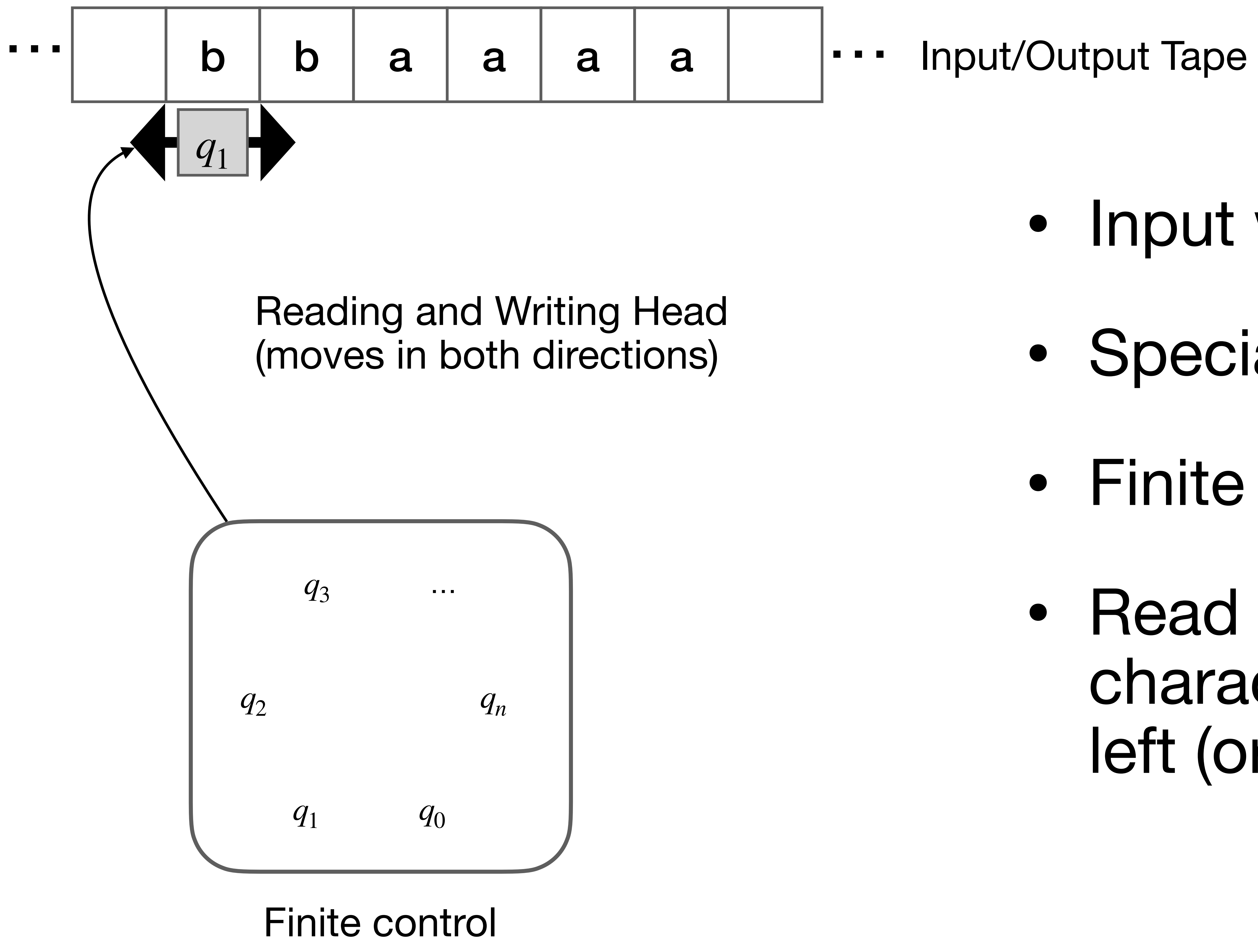
Input/Output Tape

- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).



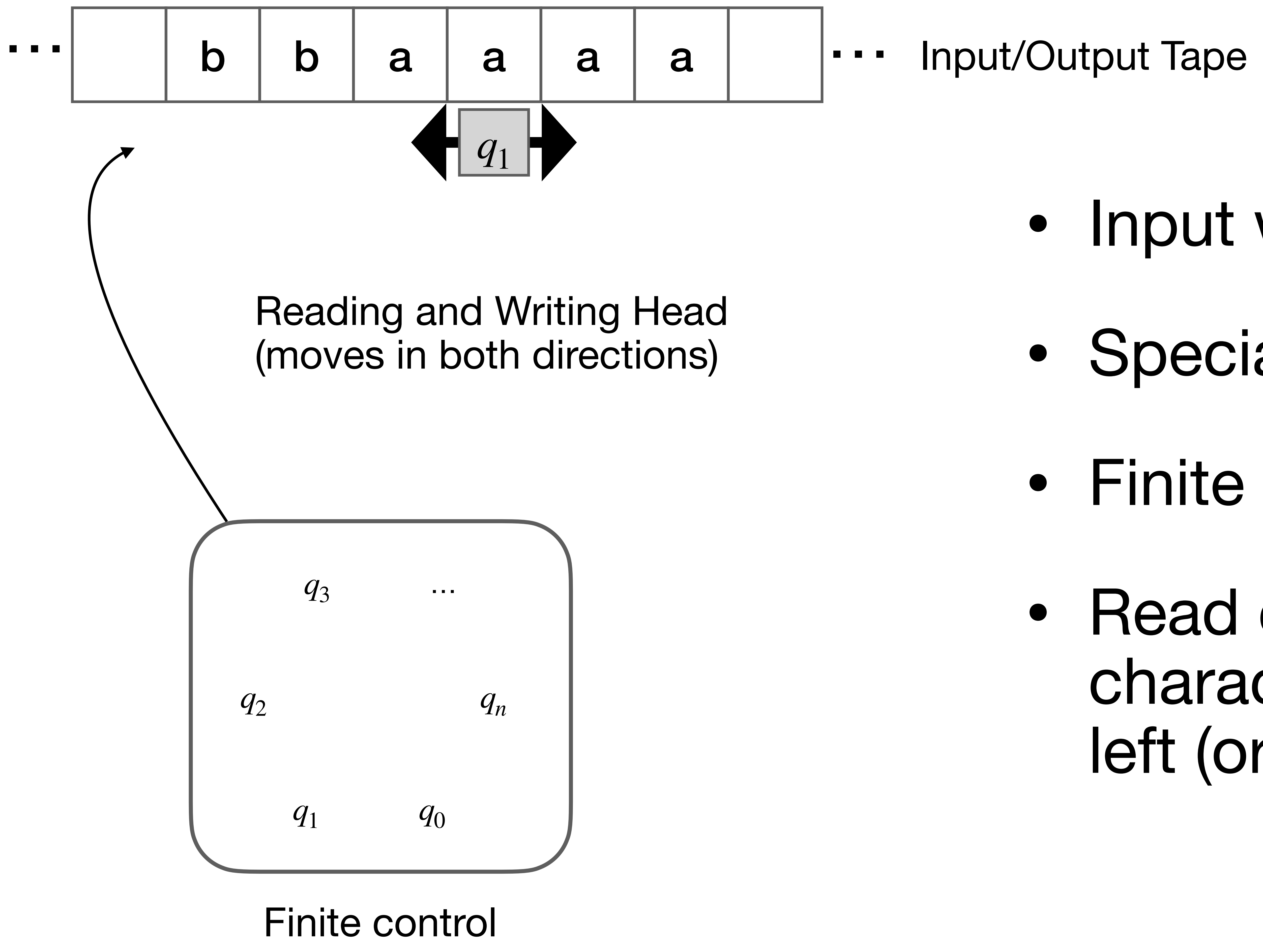
Finite control

Turing Machine



- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Read character under head, write character out, move the head right or left (or stay).

Turing Machine



- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Read character under head, write character out, move the head right or left (or stay).

Turing Machine
High level goals

- Church-Turing Thesis: TMs are the most general computing devices. So far, no “counter-example.”

Turing Machine

High level goals

Not a theorem

- Church-Turing Thesis: TMs are the most general computing devices. So far, no "counter-example."
- Every TM can be represented as a string.

Turing Machine

High level goals

- Church-Turing Thesis: TMs are the most general computing devices. So far, no "counter-example."
- Every TM can be represented as a string.
- The existence of a Universal Turing Machine, which is the model/inspiration for stored program computing. UTM can simulate any TM!

Turing Machine

High level goals

- Church-Turing Thesis: TMs are the most general computing devices. So far, no “counter-example.”
- Every TM can be represented as a string.
- The existence of a Universal Turing Machine, which is the model/inspiration for stored program computing. UTM can simulate any TM!
- Implications for what can be computed and what cannot be computed

Example

Computers exist because we are lazy… so Stanford's CS 103 to the rescue.

Turing Machine

Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet

Turing Machine

Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet
- Γ is a finite set called the tape alphabet

Turing Machine
Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet
- Γ is a finite set called the tape alphabet
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$: Transition function.

Turing Machine

Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet
- Γ is a finite set called the tape alphabet
- δ: $Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$: Transition function.
- $q_0 \in Q$ is called the initial state.

Turing Machine

Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet
- Γ is a finite set called the tape alphabet
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$: Transition function.
- $q_0 \in Q$ is called the initial state.
- $q_{acc} \in Q$, $q_{rej} \in Q$ are the accepting state and rejecting state, respectively.

Turing Machine

Formal definition

A Turing machine is a 7-tuple $\left(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}\right)$ where

- Q is a finite set of states
- Σ is a finite set called the input alphabet
- Γ is a finite set called the tape alphabet
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$: Transition function.
- $q_0 \in Q$ is called the initial state.
- $q_{acc} \in Q$, $q_{rej} \in Q$ are the accepting state and rejecting state, respectively.
- $\sqcup$ a special symbol for blank on the tape

Turing Machine

Transition function

current symbol on tape

new symbol on tape

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

current state

new state

direction to move head one step.

Turing Machine

Transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

current
state

Turing Machine
Transition function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

current state

symbol scanned

new state

symbol to write

Direction to move on tape

Turing Machine

Transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

current state — symbol scanned — new state — symbol to write — Direction to move on tape

From state q, on reading a:

- go to state p
- write b
- move head *Left*
- missing transitions lead to BSOD

Turing Machine

Transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

current state — symbol scanned — new state — symbol to write — Direction to move on tape

From state q, on reading a:

- go to state p
- write b
- move head *Left*
- missing transitions lead to BSOD

$$\delta(q, a) = (p, b, L)$$

aka CRASH!!

More example(s)

Same link as last time again

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.
  - Need to distinguish cases, related to “deciders” (partial and total).

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.
  - Need to distinguish cases, related to “deciders” (partial and total).
- *Recursively enumerable* (aka RE) languages

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.
  - Need to distinguish cases, related to “deciders” (partial and total).
- *Recursively enumerable* (aka RE) languages
  - $L = \{L(M) \mid M \text{ some Turing machine}\}$

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.
  - Need to distinguish cases, related to “deciders” (partial and total).
- *Recursively enumerable* (aka RE) languages
  - $L = \{L(M) \mid M \text{ some Turing machine}\}$
- *Recursive/decidable* languages

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- TMs have a new behavior - it may never halt.
  - Need to distinguish cases, related to “deciders” (partial and total).
- *Recursively enumerable* (aka RE) languages
  - $L = \{L(M) \mid M \text{ some Turing machine}\}$
- *Recursive/decidable* languages
  - $L = \{L(M) \mid M \text{ some Turing machine that halts on all inputs}\}$

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- A **total decider** is a TM which will always halt in an accept or reject state.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- A **total decider** is a TM which will always halt in an accept or reject state.
  - *Recursive languages* are called decidable language precisely because they have total deciders.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- A **total decider** is a TM which will always halt in an accept or reject state.
    - *Recursive languages* are called decidable language precisely because they have total deciders.
- A **partial decider** is a TM, which if given string in its language, will reach an accept state.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- A **total decider** is a TM which will always halt in an accept or reject state.
    - *Recursive languages* are called decidable language precisely because they have total deciders.
- A **partial decider** is a TM, which if given string in its language, will reach an accept state.
    - If given a string that is not in its language, it could loop forever.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- A **total decider** is a TM which will always halt in an accept or reject state.
  - *Recursive languages* are called decidable language precisely because they have total deciders.
- A **partial decider** is a TM, which if given string in its language, will reach an accept state.
  - If given a string that is not in its language, it could loop forever.
  - *Recursively enumerable* languages are ones for which a partial decider exists.

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?
- A ***semi-decidable*** problem (equivalent of recursively enumerable) could be:

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?

- A ***semi-decidable*** problem (equivalent of recursively enumerable) could be:
  - **Decidable** - equivalent of recursive (**TM** always accepts or rejects).

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
    - What languages are RE?
    - Which are recursive?
    - What is the difference?
    - What makes a language decidable?

- A ***semi-decidable*** problem (equivalent of recursively enumerable) could be:
    - **Decidable** - equivalent of recursive (**TM** always accepts or rejects).
    - **Undecidable** - Problem is not recursive

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?

- A ***semi-decidable*** problem (equivalent of recursively enumerable) could be:
  - **Decidable** - equivalent of recursive (**TM** always accepts or rejects).
  - **Undecidable** - Problem is not recursive
- There are also undecidable problems that are not recursively enumerable!

Languages defined by a Turing machine

Recursive vs. Recursively Enumerable

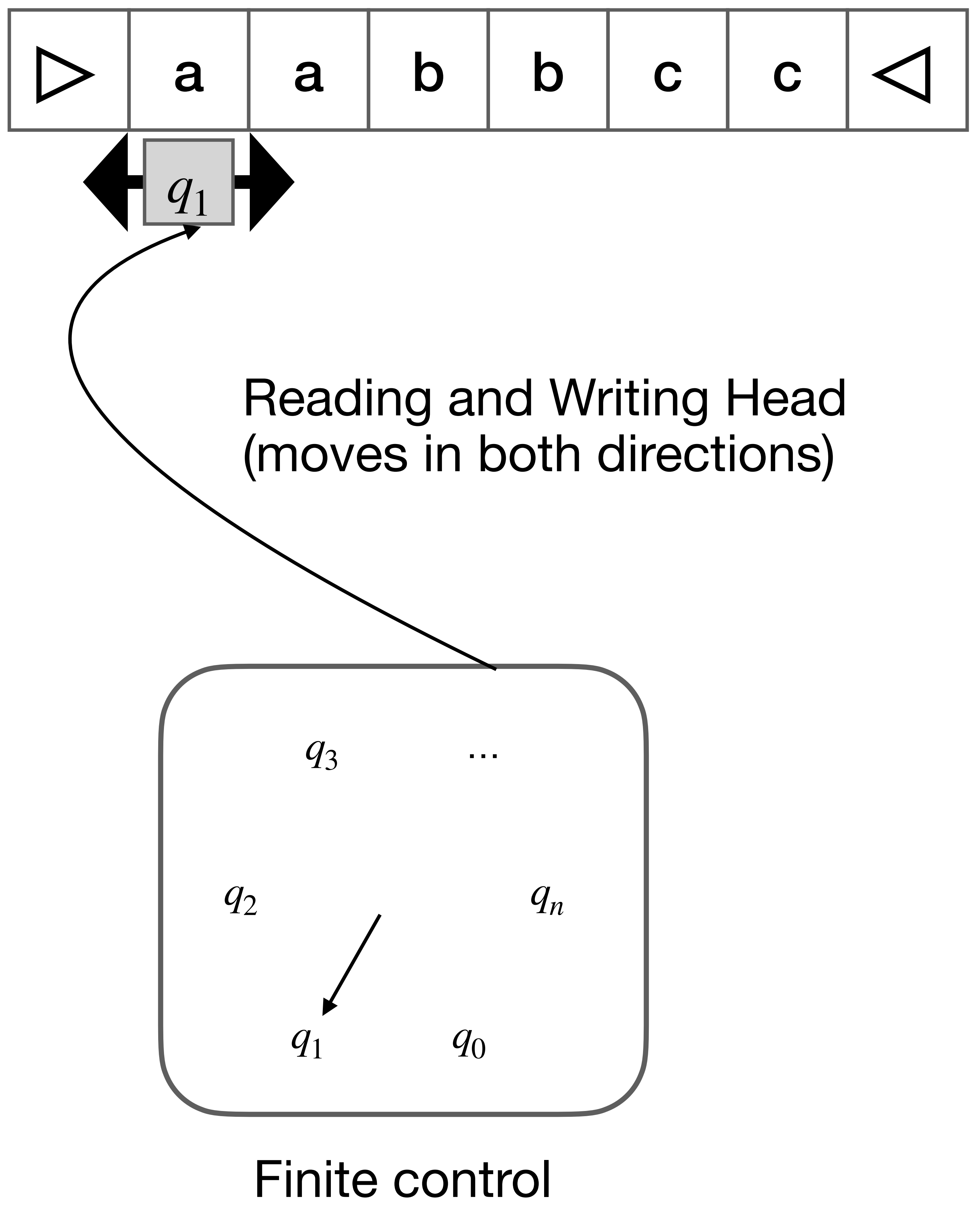
- Fundamental questions:
  - What languages are RE?
  - Which are recursive?
  - What is the difference?
  - What makes a language decidable?

More on these closer to end of semester.

- A ***semi-decidable*** problem (equivalent of recursively enumerable) could be:
  - **Decidable** - equivalent of recursive (**TM** always accepts or rejects).
  - **Undecidable** - Problem is not recursive
- There are also undecidable problems that are not recursively enumerable!

Linear Bounded Automata

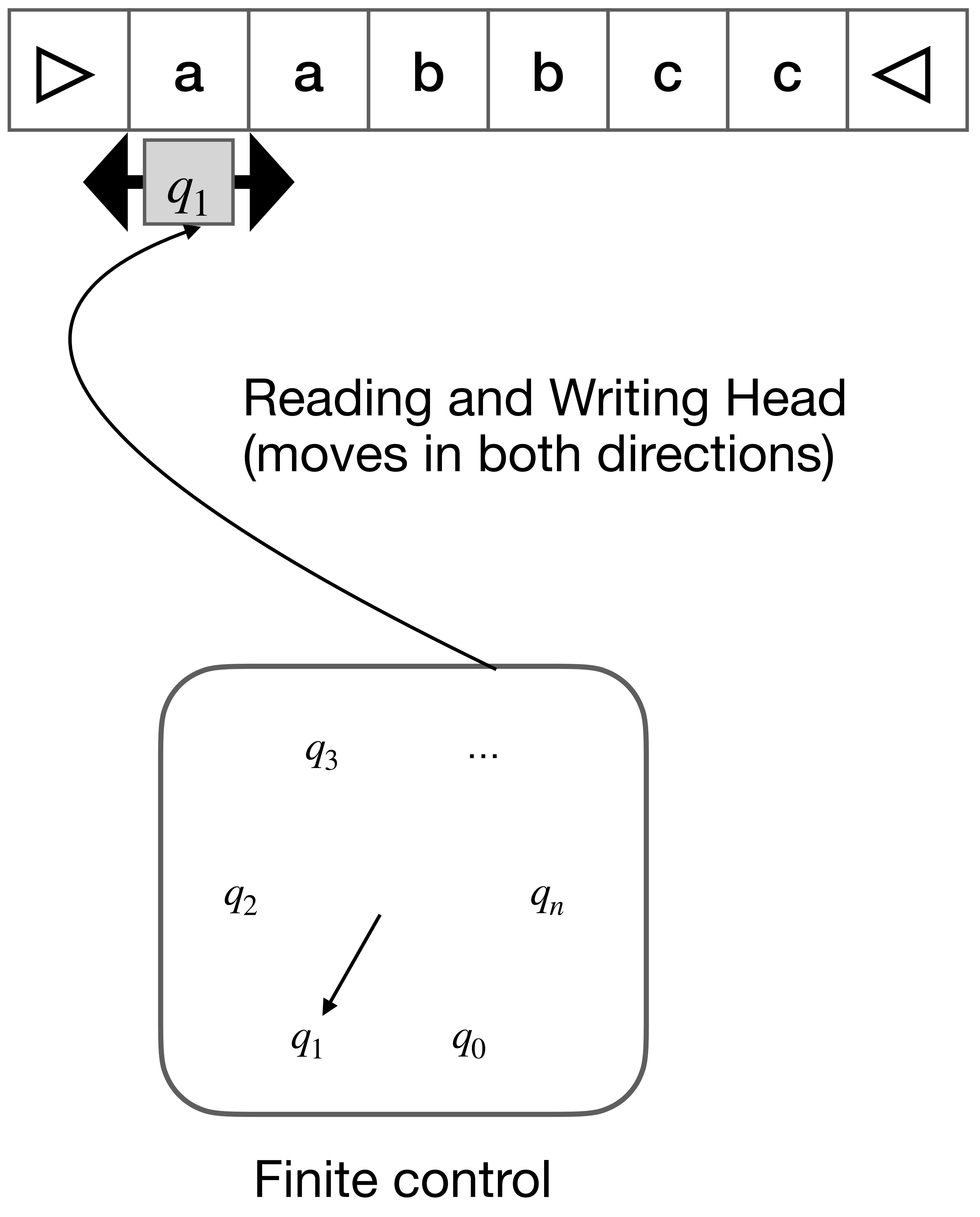
Relation to TMs



- We skipped LBAs
    - They can be thought of as *restricted* Turing Machines.

Linear Bounded Automata

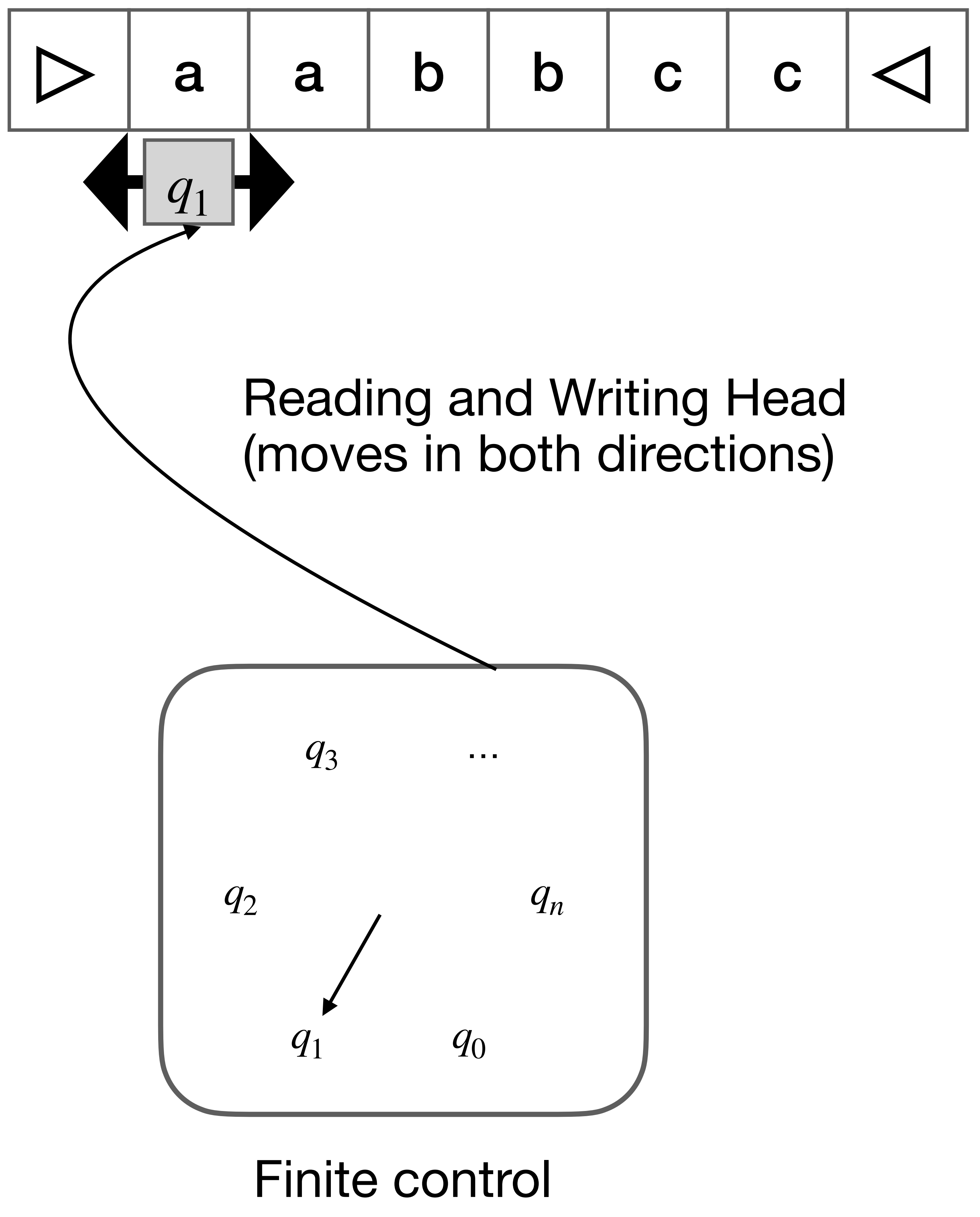
Relation to TMs



- We skipped LBAs
    - They can be thought of as *restricted* Turing Machines.
    - Tape used grows *linearly* with size of input

Linear Bounded Automata

Relation to TMs



- We skipped LBAs
  - They can be thought of as *restricted* Turing Machines.
  - Tape used grows *linearly* with size of input
- (Nondeterministic) LBA can recognize all context-sensitive languages.

Wrap up …

… the four-week tour of Models of Computation.

Chomsky's terminology

Grammar	Languages	Production Rules	Automation	Examples
Type-0	Turing ~~machine~~ recognizable	$\gamma \rightarrow \alpha$ (no constraints)	Turing Machines	L = { w | w is a TM which halts }
Type-1	Context-sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Linear Bounded Automata	$L = \{ a^n b^n c^n \mid n > 0 \}$
Type-2	Context-free	$A \rightarrow \alpha$	Pushdown Automata	$L = \{ a^n b^n \mid n > 0 \}$
Type-3	Regular	$A \rightarrow aB$	Non-determinstic Finite Automata	$L = \{ a^n \mid n > 0 \}$

same as DFAs or RegExes

Next class

Universal Turing Machines

Next class

Universal Turing Machines

- **Theorem (Turing, 1936):** There is a Turing Machine, called the Universal Turing Machine, that, when run on an input of the form $\langle M, w \rangle$ where M is a TM and w is a string, simulates M running on w and does whatever M does on w — accepts, rejects, loops.

Next class

Universal Turing Machines

- **Theorem (Turing, 1936):** There is a Turing Machine, called the Universal Turing Machine, that, when run on an input of the form $\langle M, w \rangle$ where M is a TM and w is a string, simulates M running on w and does whatever M does on w — accepts, rejects, loops.

- Question for the weekend: Recall that the language of a TM is the set of strings it accepts.

Next class

Universal Turing Machines

- **Theorem (Turing, 1936):** There is a Turing Machine, called the Universal Turing Machine, that, when run on an input of the form $\langle M, w \rangle$ where M is a TM and w is a string, simulates M running on w and does whatever M does on w — accepts, rejects, loops.
- Question for the weekend: Recall that the language of a TM is the set of strings it accepts.

What is the language of a UTM?