Turing Machine I

Zeyu Mi

2023-12-13

Adapted From:
《数理逻辑与集合论》11.2
IALC 1.1,1.5,2.2,8.2

Review

- **Function**
  - Definition and common functions.
  - Injective, surjective, bijective.

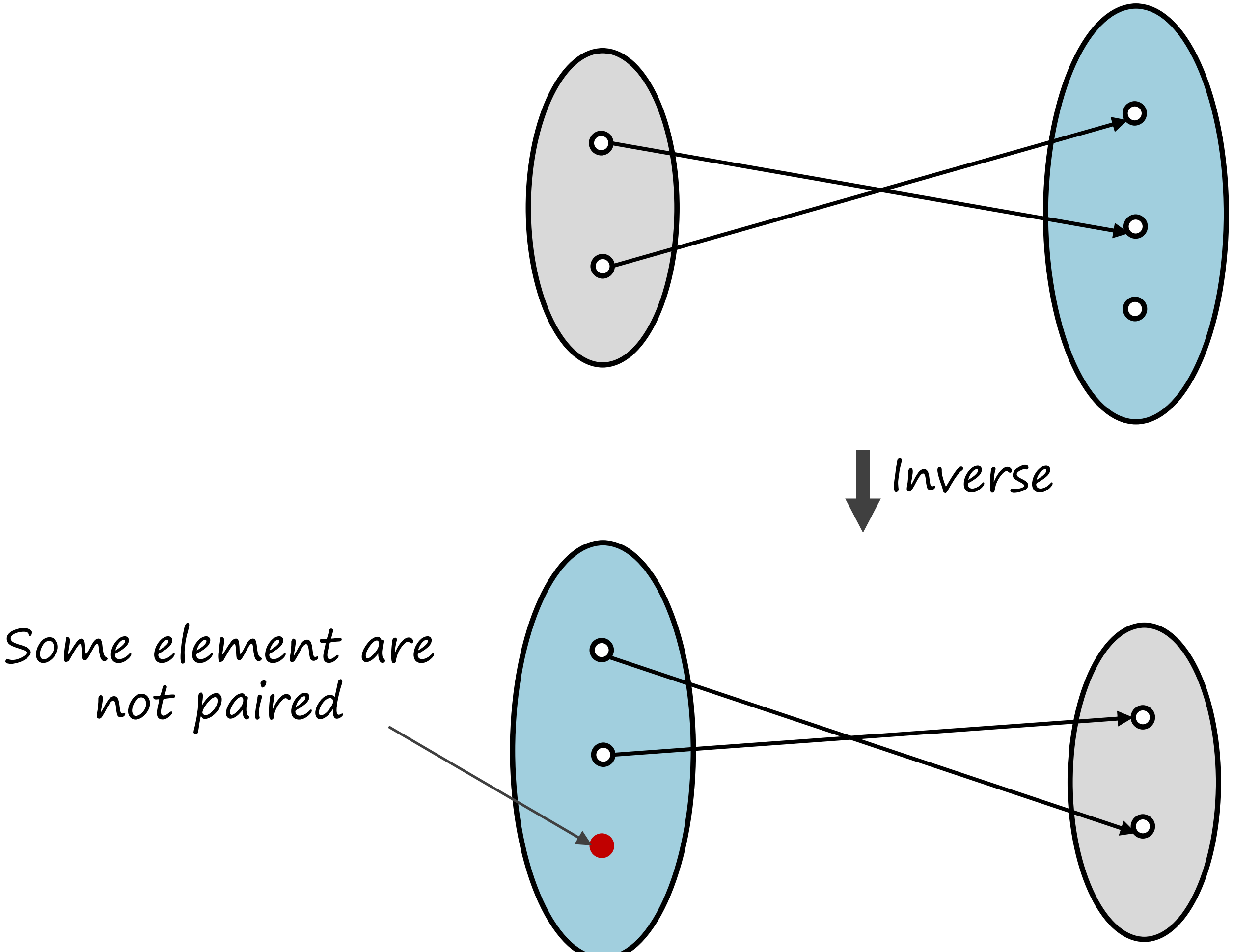
Review: Axiom of Choice

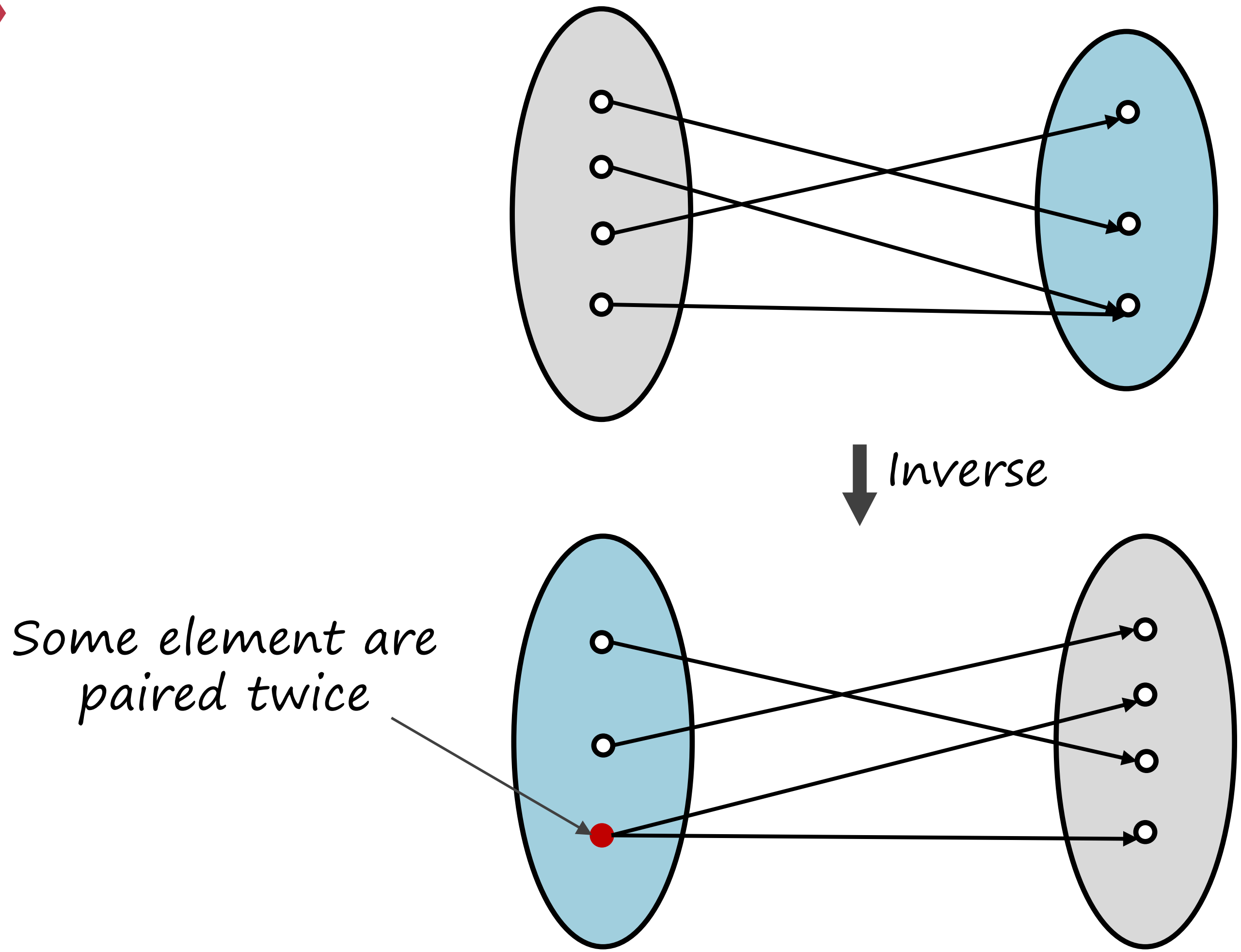
ZF.10 Axiom of choice(选择公理)

For any relation R, there exists a function F, such that F is a subset of R and the domain of F and R are equal.

$$(\forall \text{ Relation } R)(\exists \text{ Function } F)(F \subseteq R \wedge dom(R) = dom(F))$$

-
-
-


Inverse
Some element are
not paired


Inverse
Some element are
paired twice

Inverse of Function

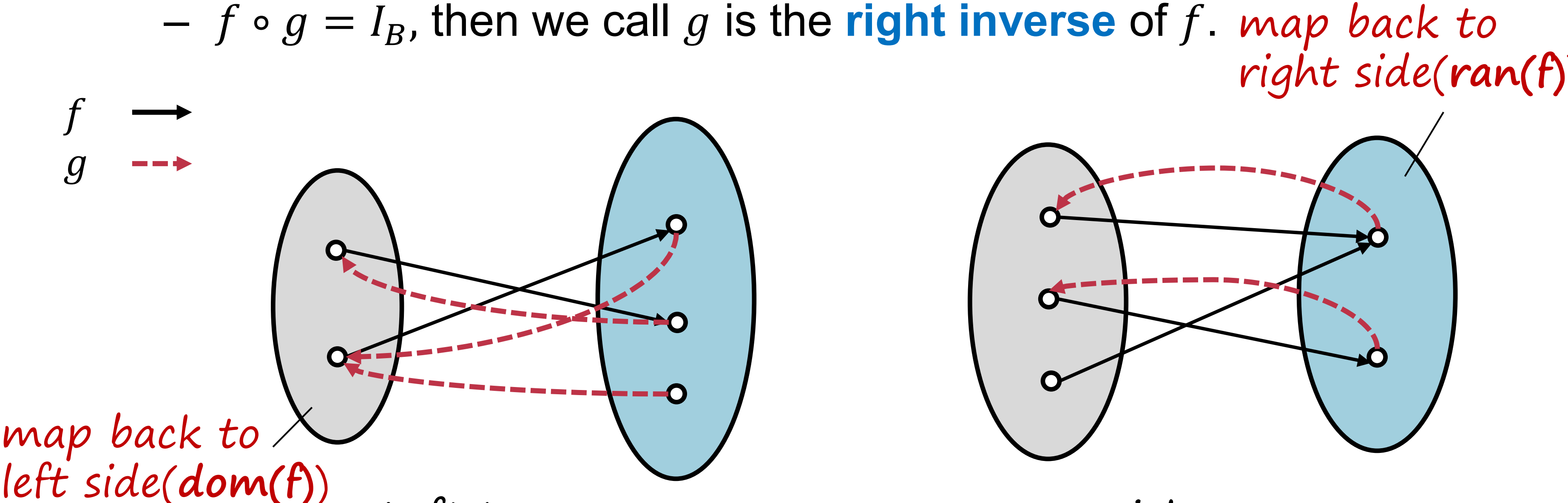
- Let $f: \boldsymbol{A} \to \boldsymbol{B}$
  - The inverse (defined by relation) of f may not be a function
  - If the inverse of f is a function, we call f is **inversible** （可逆的）

$f: A \to B$, if there exists $g: B \to A$, such that $g \circ f = I_A$ and $f \circ g = I_B$, we call that f is **inversible**, and g is the **inverse function(反函数)** of f.

Some times $g \circ f = I_A$ and $f \circ g = I_B$ may not hold simultaneously.

Left Inverse(左逆) and Right Inverse(右逆)

- Let $f: \boldsymbol{A} \to \boldsymbol{B}$, if there exists $g: \boldsymbol{B} \to \boldsymbol{A}$, such that
  - $g \circ f = I_A$, then we call g is the **left inverse** of f.
  - $f \circ g = I_B$, then we call g is the **right inverse** of f.

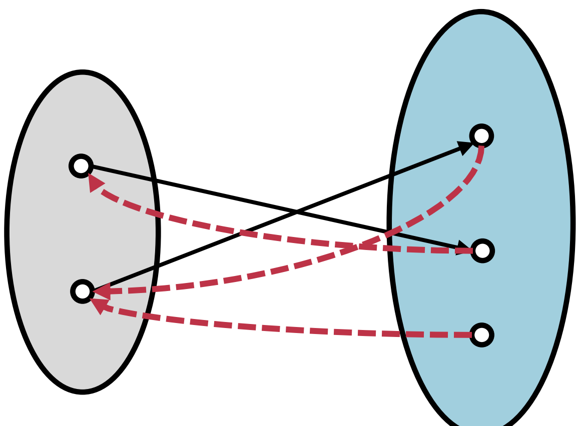


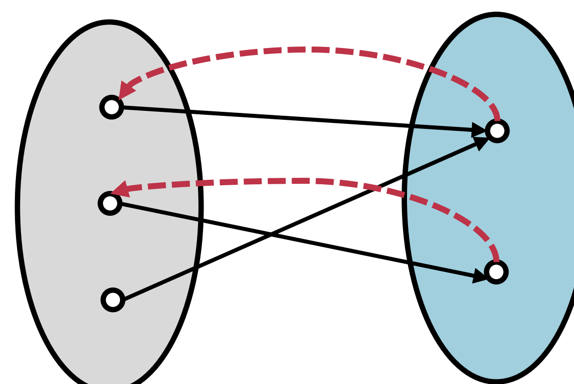
Left Inverse
$g \circ f = I_A$

Right Inverse
$f \circ g = I_B$

Left Inverse and Right Inverse

- Let $f\colon \boldsymbol{A} \to \boldsymbol{B}$, $A \neq \emptyset$
  - f has left inverse if and only if f is injective.
  - f has right inverse if and only if f is surjective.
  - f has both left and right inverse if and only if f is bijective.
  - f is bijective, then the left and right inverse of f are equal.





Left Inverse and Right Inverse

- Let $f\colon \boldsymbol{A} \to \boldsymbol{B}$, $A \neq \emptyset$, assume that g is the left inverse of f, and h is the right inverse of f.
  - We have $g \circ f = I_A$, $f \circ h = I_B$
  - Then $g = g \circ I_B = g \circ (f \circ h) = (g \circ f) \circ h = I_A \circ h = h$
  - So if f has both left and right inverse, then they are equal.

Inverse of Function

- Let $f: \boldsymbol{A} \to \boldsymbol{B}$
  - The inverse (defined by relation) of f may not be a function
  - If the inverse of f is a function, we call f is **inversible** （可逆的）

$f: A \to B$, if there exists $g: B \to A$, such that $g \circ f = I_A$ and $f \circ g = I_B$, we call that f is **inversible**, and g is the **inverse function(反函数)** of f.

- f has both left and right inverse if and only if f is bijective.

⬇

f is inversible if and only if f is bijective.

Inverse of Function

- Let $f: \boldsymbol{A} \rightarrow \boldsymbol{B}$ be a bijection
  - f^{-1} is a function and is the inverse function of f.
  - f^{-1} is bijective too.
  - $\forall x \in A, f^{-1}\left(f(x)\right) = x$. $\forall y \in B, f\left(f^{-1}(y)\right) = y$

How we investigate computability?

2.1 Discussion on Problems

Part1. Intro & Set theory(I) : Basics & Formal Language.

Part2. Set Theory(II): Axiom system & Cardinality.

Part3. Capture Structures: Binary Relation & Function

2.2 Discussion on Computation

Part1. Turing Machine Basics.

Part2. Variants of Turing Machine. Church-Turing Thesis.

2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

Ultimate Problem:

What problems can a computer solve?

What We Have Done

- **Computational Problem(计算问题)**
  - Well-defined input
  - Constraints that the output must satisfy.
- **A computational problem is computable(可计算) if there is a procedure (or algorithm) for it which:**
  - Rigorously follows some instructions, requires no ingenuity.
  - Always finishes (terminates) after a finite number of steps.

What We Have Done

- **A computational problem can be converted into language recognizing problem.**
  - Set basics & formal language theory
  - A language is naturally a set of string.
  - Given a language L and take w as input, determine whether $w \in L$.

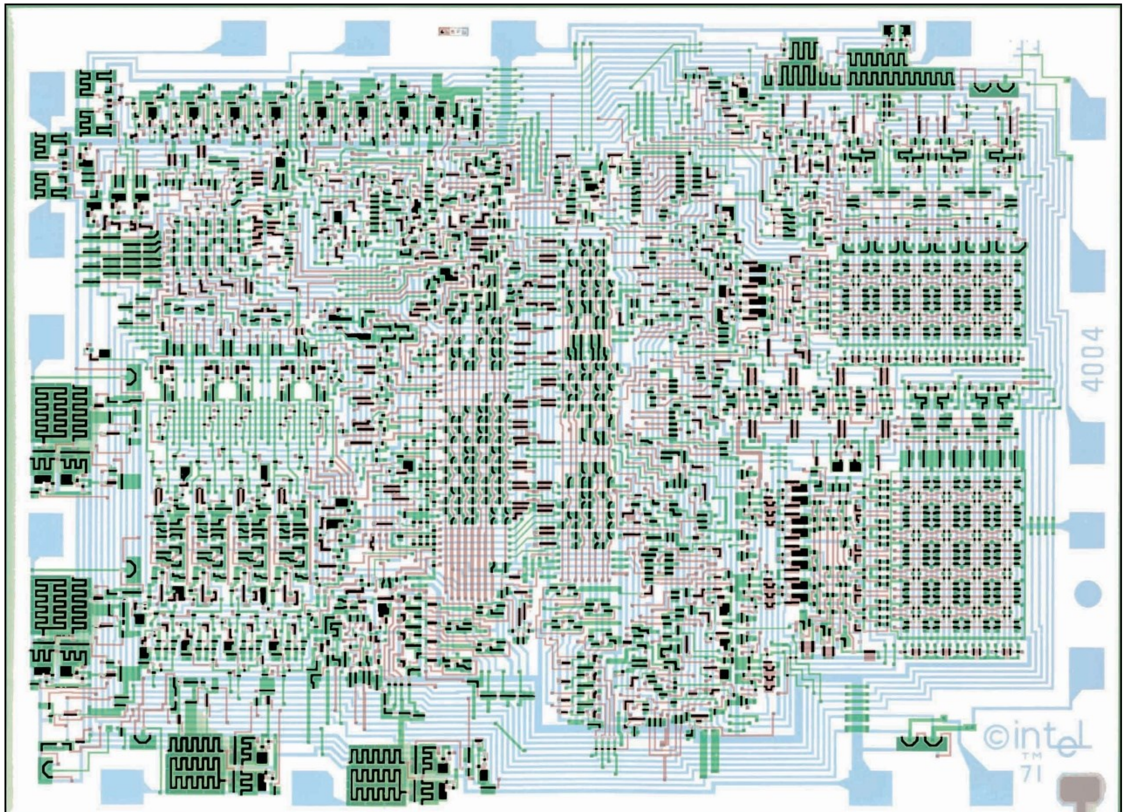
~~What problems can a computer solve?~~

↓

What languages can a computer recognize?

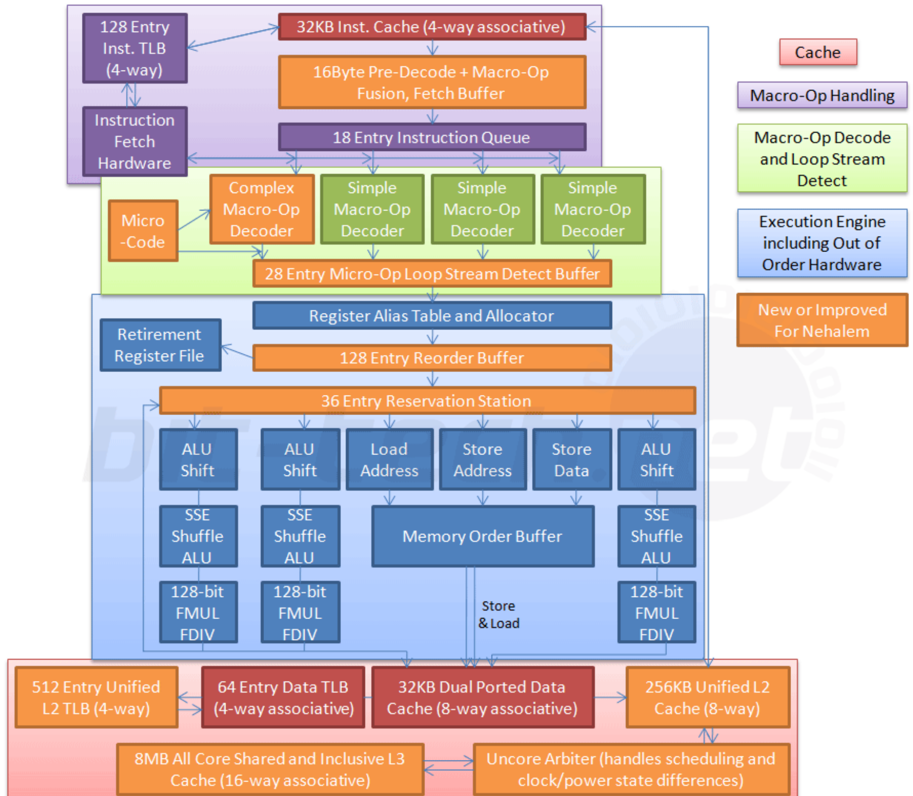
What languages can a **computer** recognize?

Computers are complicated.



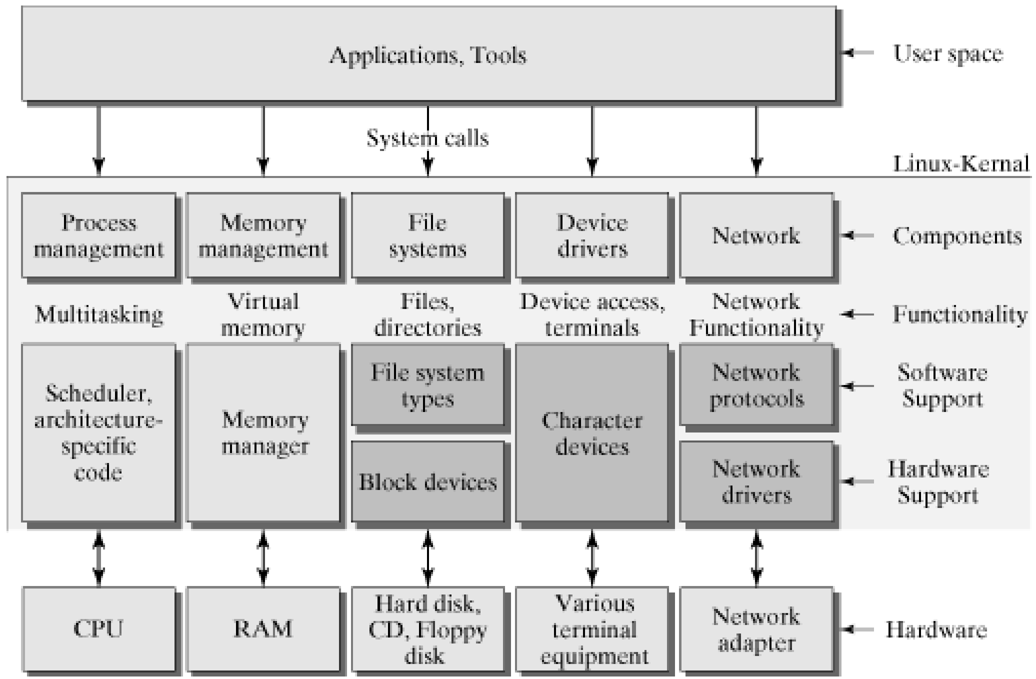
Intel 4004, first commercial CPU

Computers are complicated.



Core i7 Architecture.

Computers are complicated.



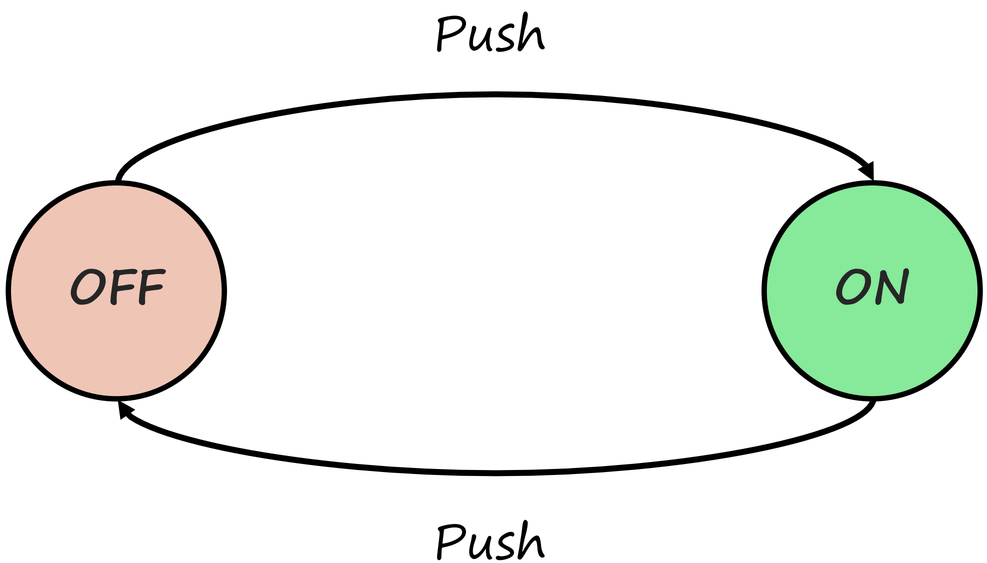
Architecture of Operating System

We need to model computer in a simple and clean manner!!

Automata(自动机) Theory

- An automaton is a mathematical model of a computing device
  - It is significantly easier to manipulate our abstract models of computers than it is to manipulate actual computers
  - If we pick our models correctly, we can make broad, sweeping claims about huge classes of real computers by arguing that they're just special cases of our more general models.

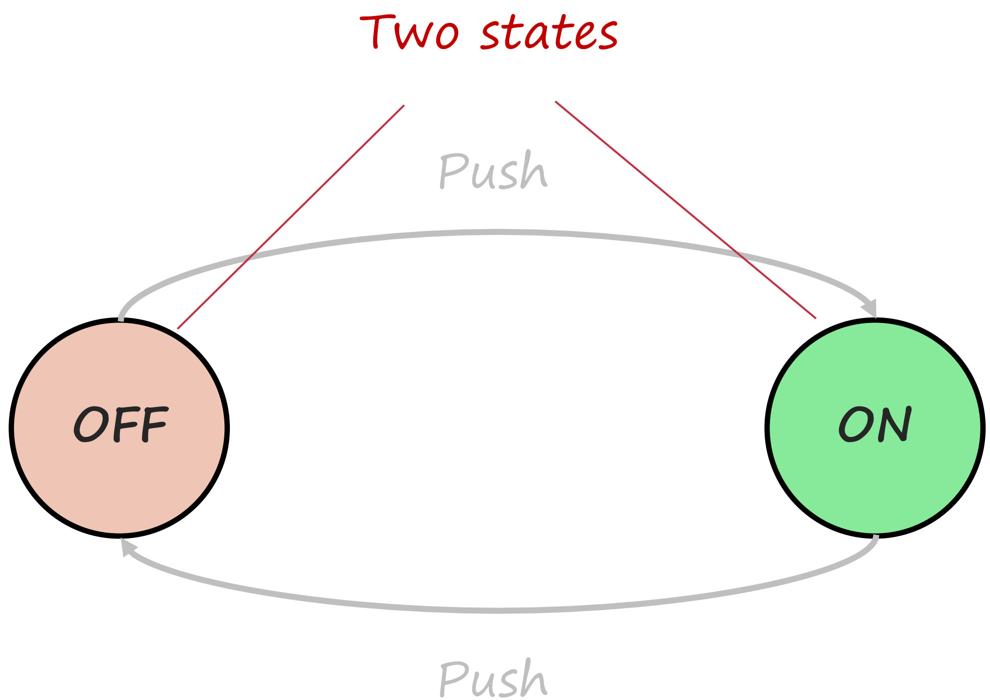
What is an Automaton



An "automata" for an on-off switch

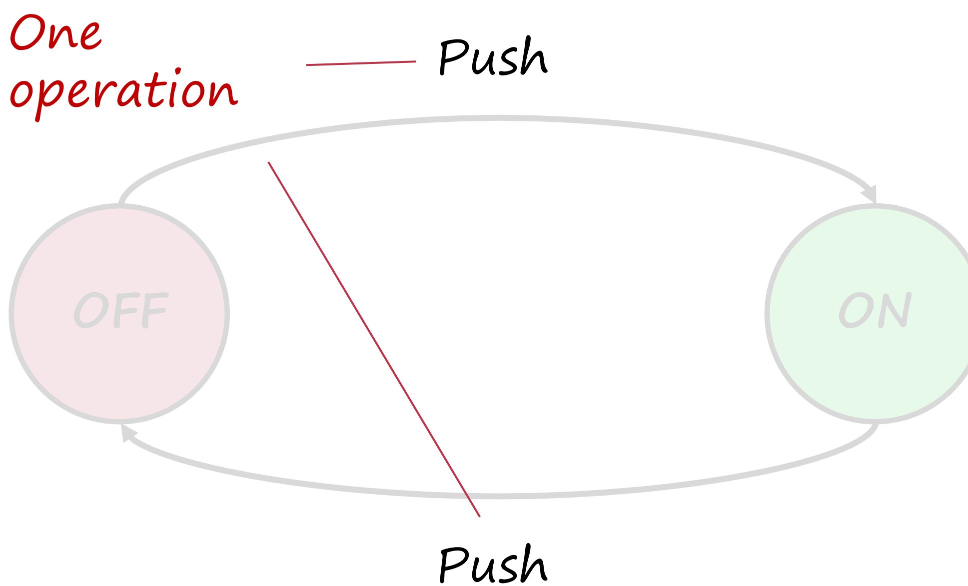
What is an Automaton

- Q: {“off”, “on”}



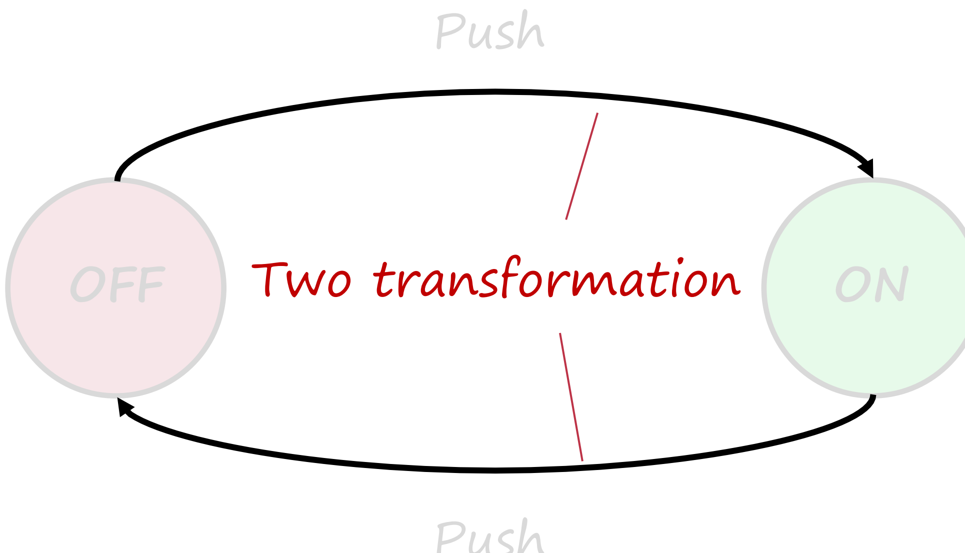
What is an Automaton

- Q: {“off”, “on”}
- Σ : {“push”}

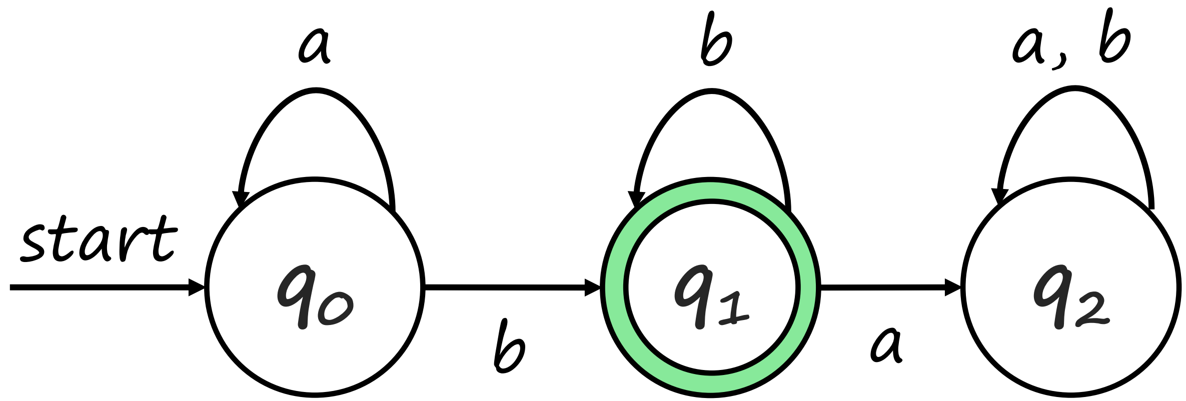


What is an Automaton

- Q: {"off", "on"}
- Σ : {"push"}
- δ: {("on", "push", "off"), ("off", "push", "on")}



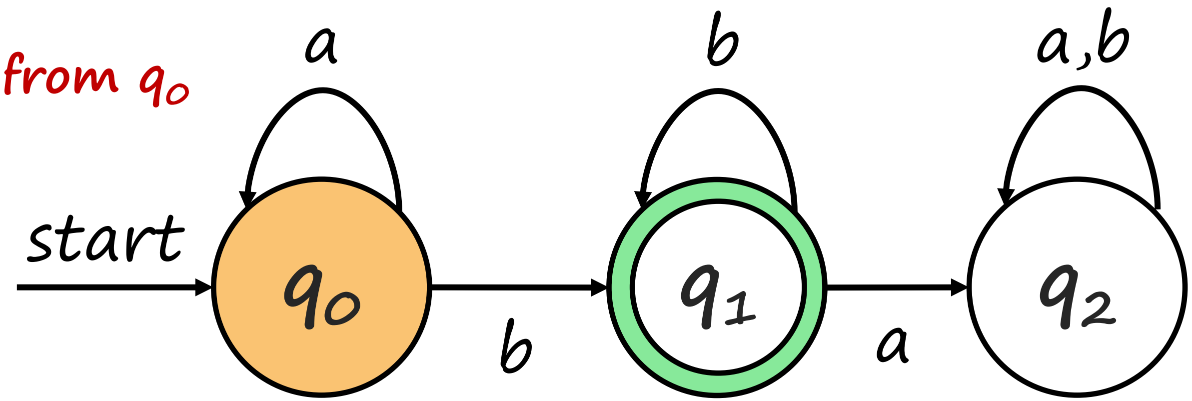
What is an Automaton



A automata for “aa…bb…”

What is an Automaton

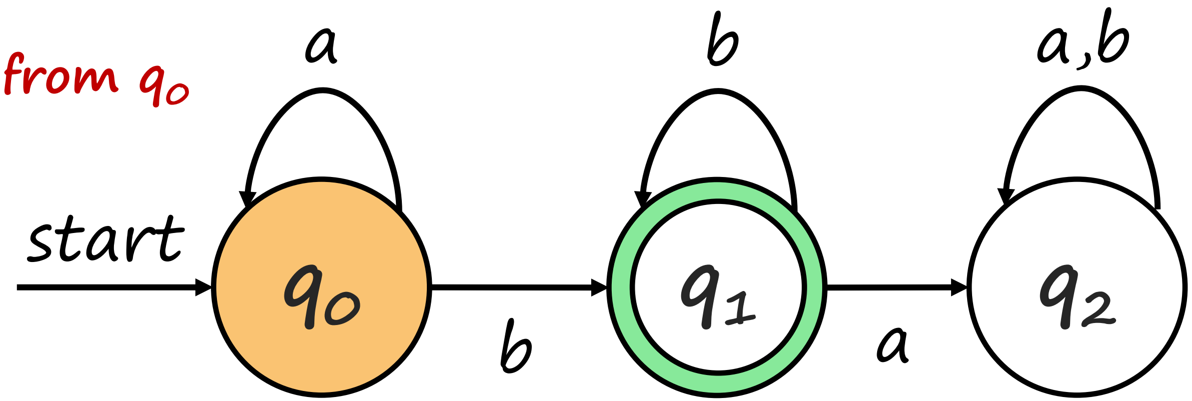
We'll start from q_0



Input: a a b b

What is an Automaton

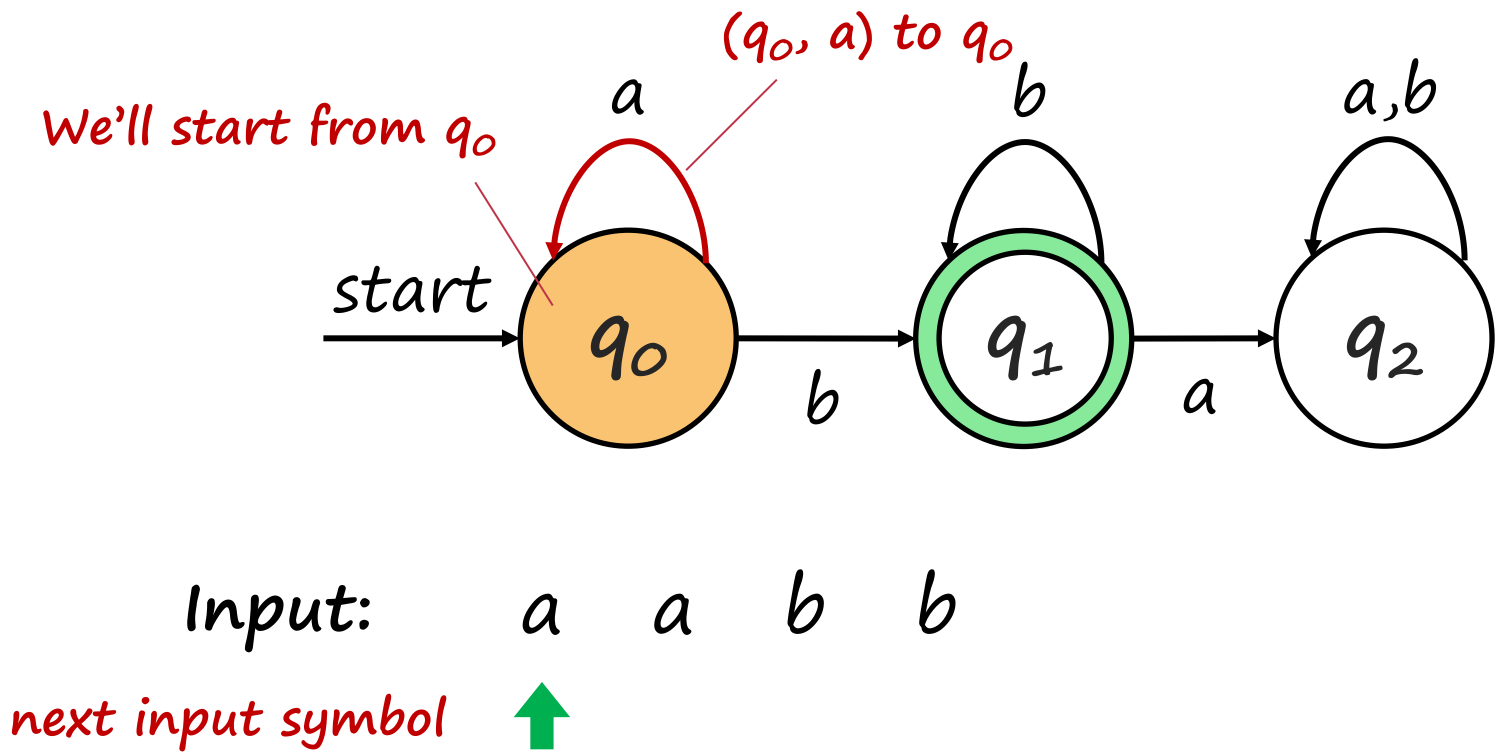
We'll start from q_0



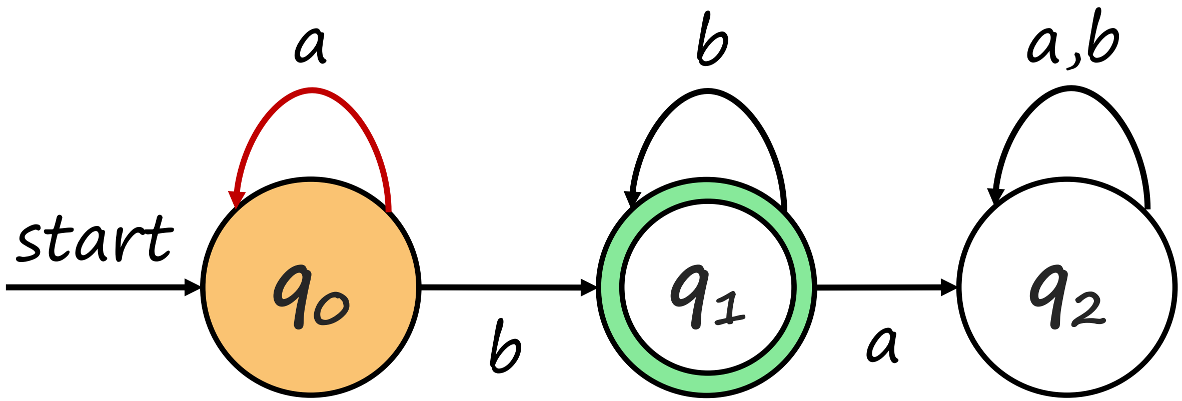
Input: a a b b

next input symbol

What is an Automaton



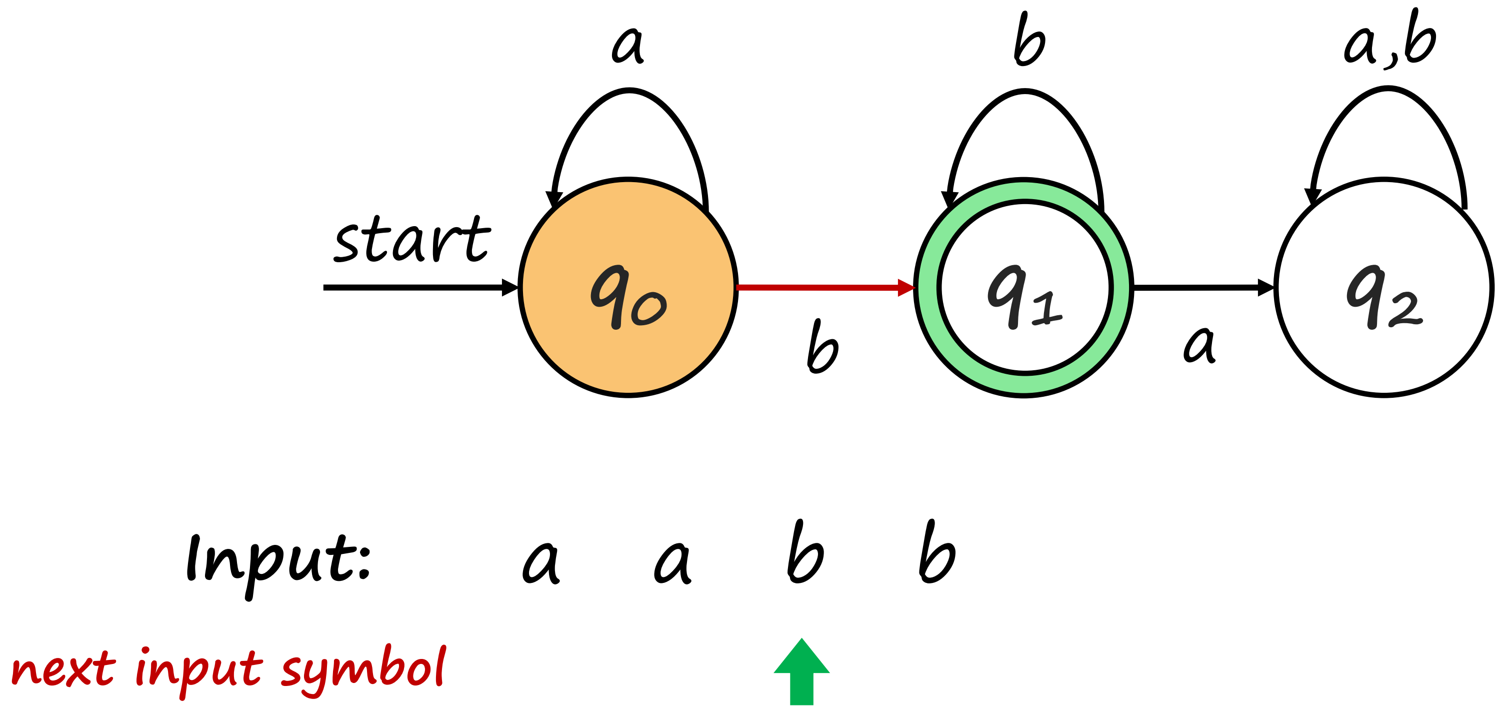
What is an Automaton



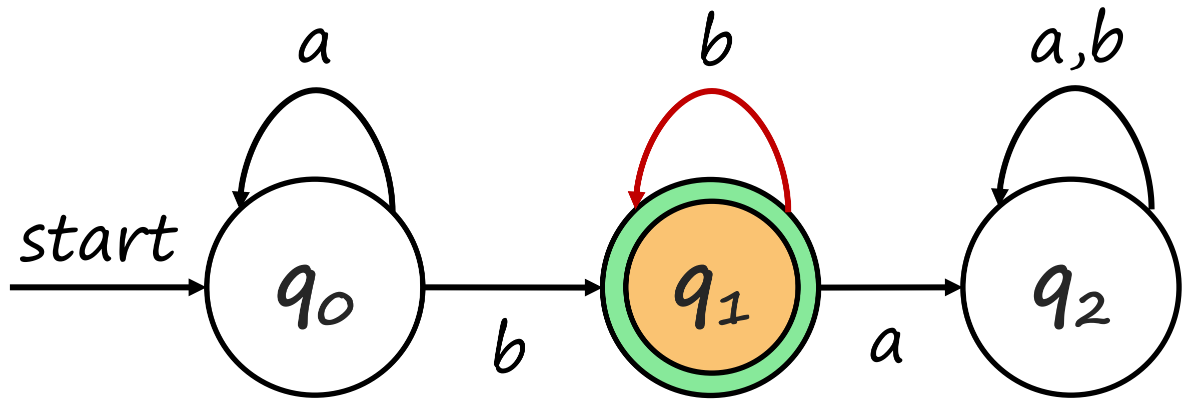
Input: a a b b

next input symbol

What is an Automaton



What is an Automaton

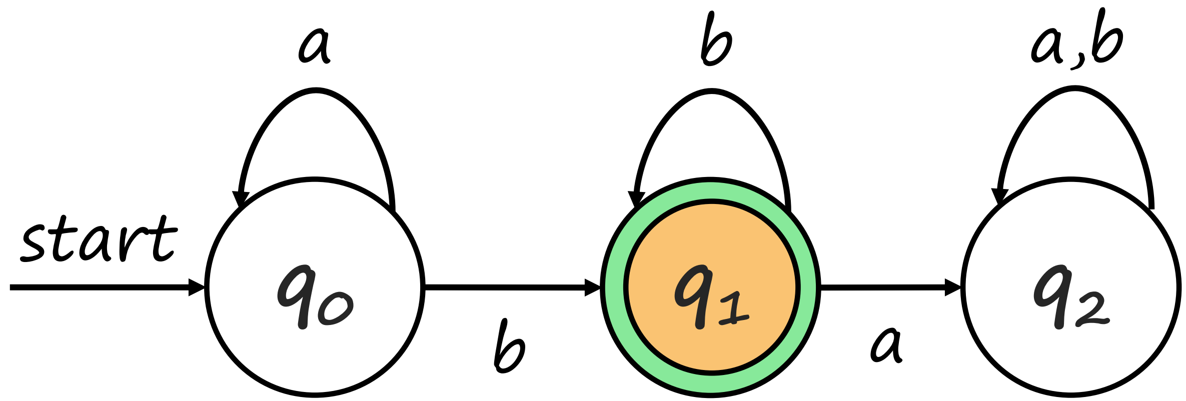


Input: a a b b

next input symbol

What is an Automaton

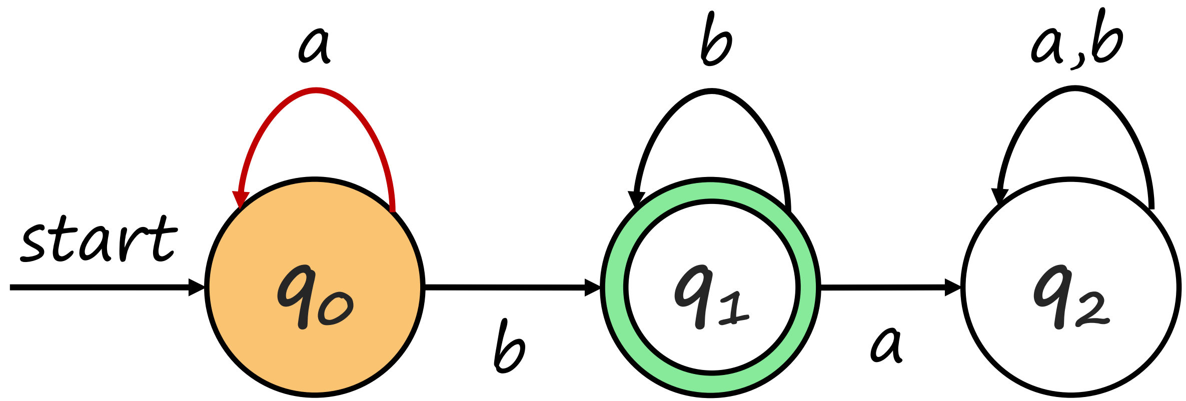
q_1 is the final state.



Input: a a b b

next input symbol

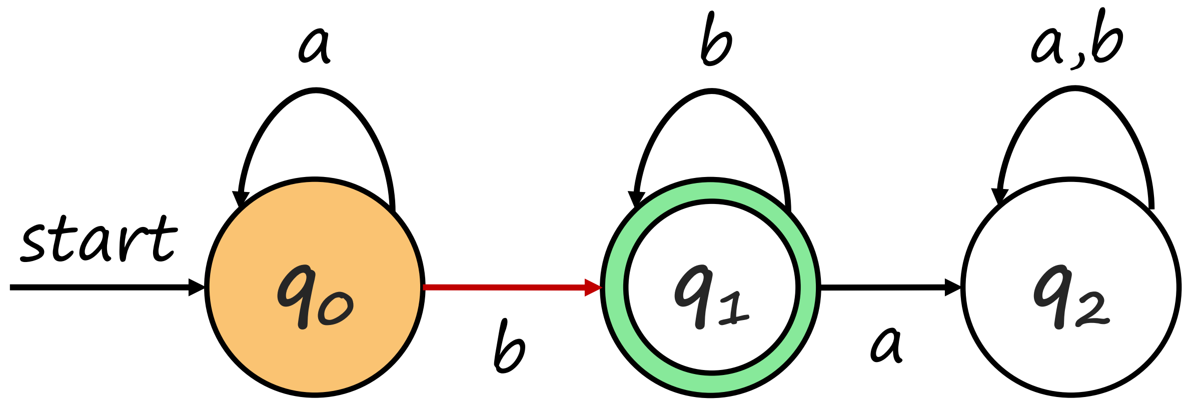
What is an Automaton



Input: a b b a

next input symbol

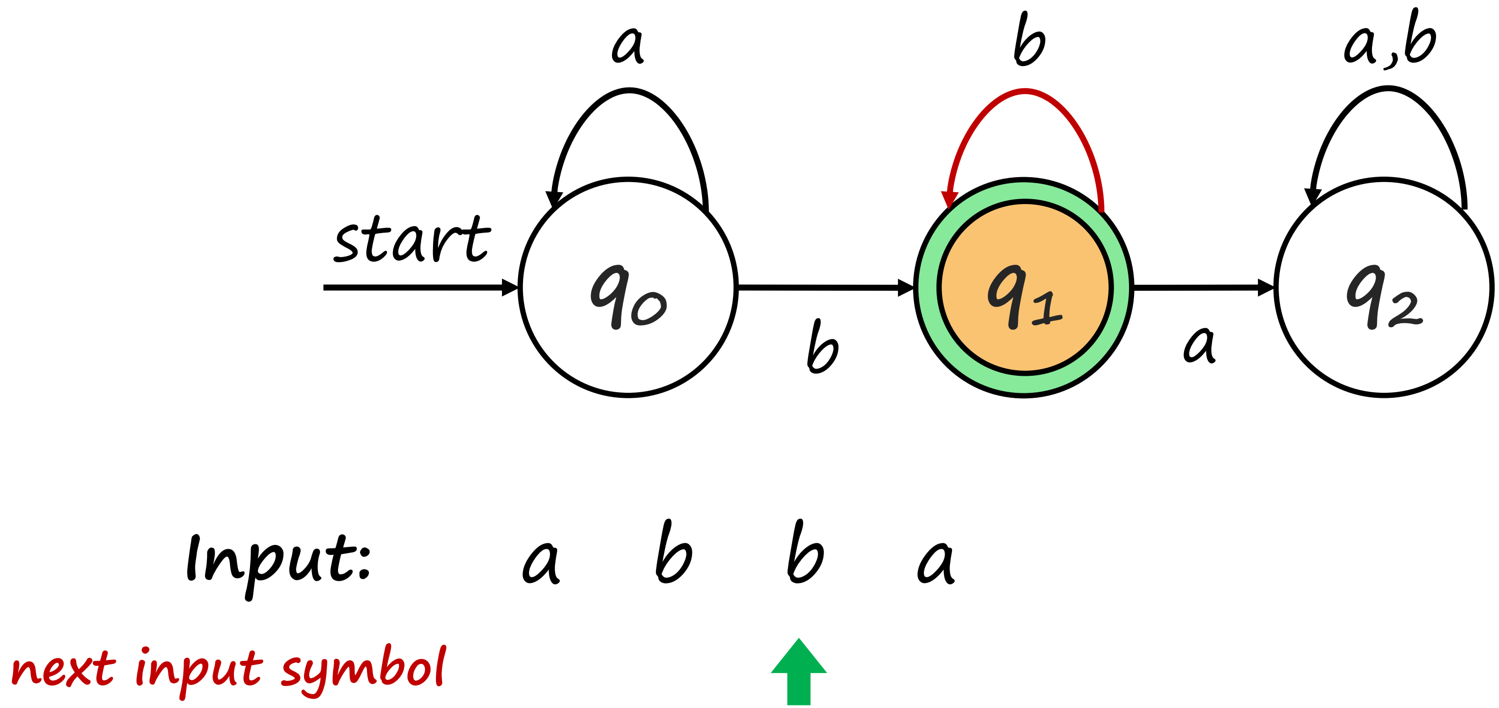
What is an Automaton



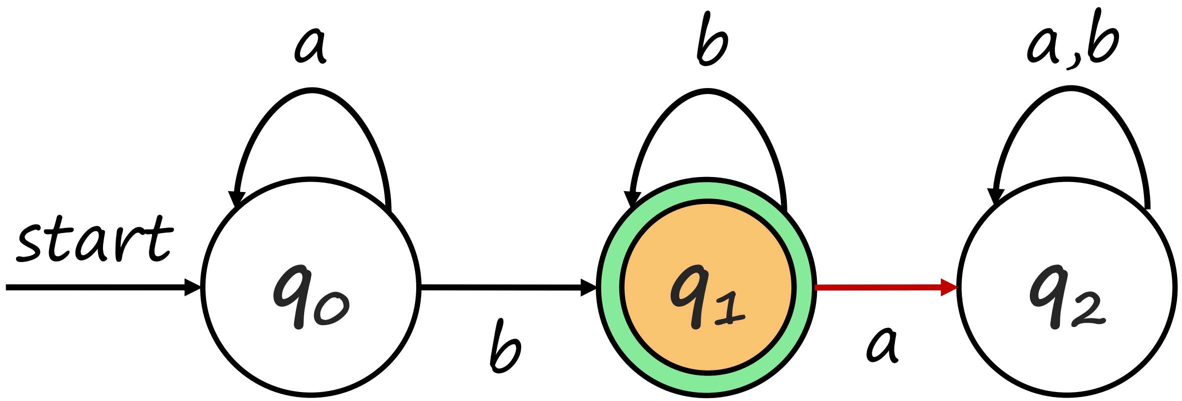
Input: a b b a

next input symbol

What is an Automaton



What is an Automaton

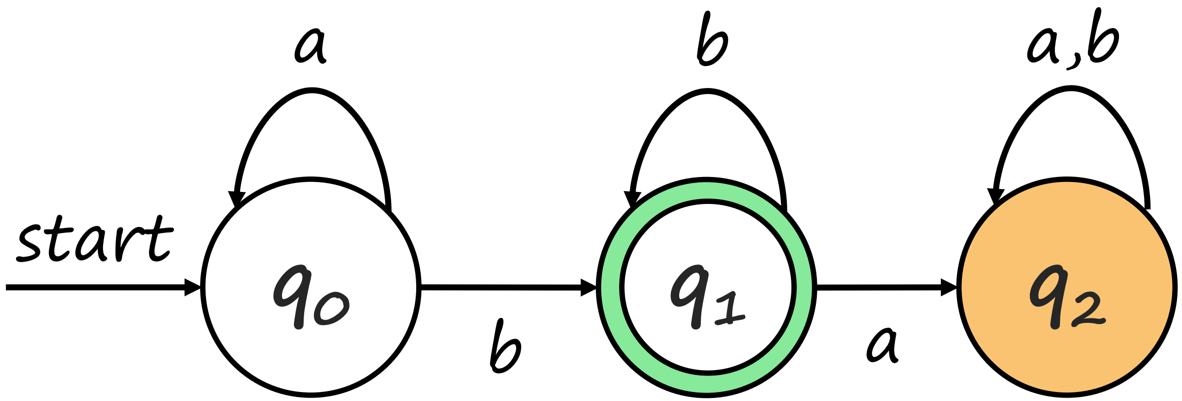


Input: a b b a

next input symbol

What is an Automaton

q_2 is not the final state.

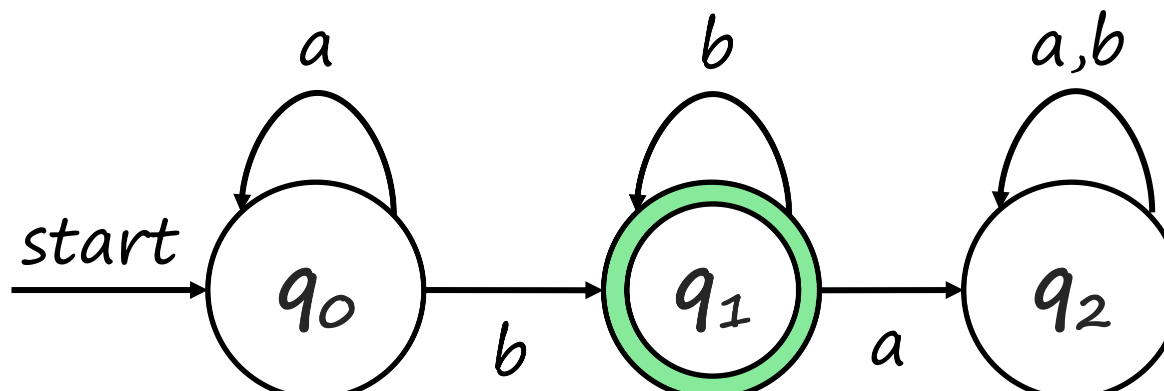


Input: a b b a

next input symbol

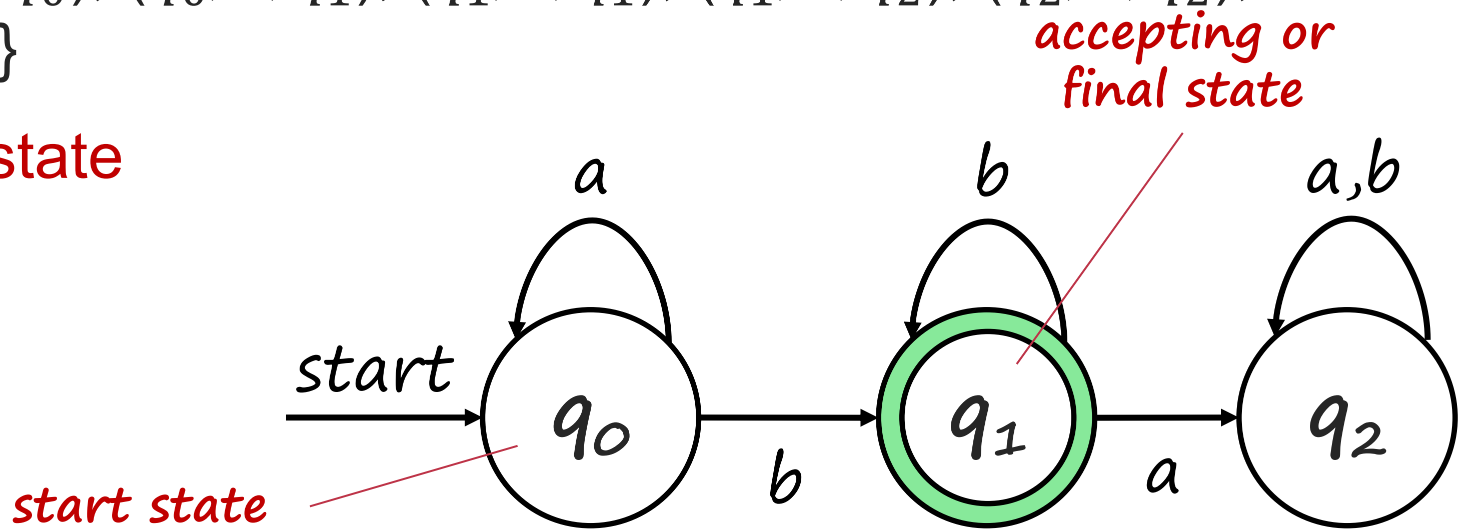
What is an Automaton

- Q: $\{q_0, q_1, q_2\}$
- Σ : $\{a, b\}$
- δ: $\{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$



What is an Automaton

- Q: $\{q_0, q_1, q_2\}$
- Σ : $\{a, b\}$
- δ: $\{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_1), (q_1, a, q_2), (q_2, a, q_2), (q_2, b, q_2)\}$
- q_0: start state
- F: $\{q_1\}$



Deterministic Finite Automaton(DFA, 确定有限自动机)

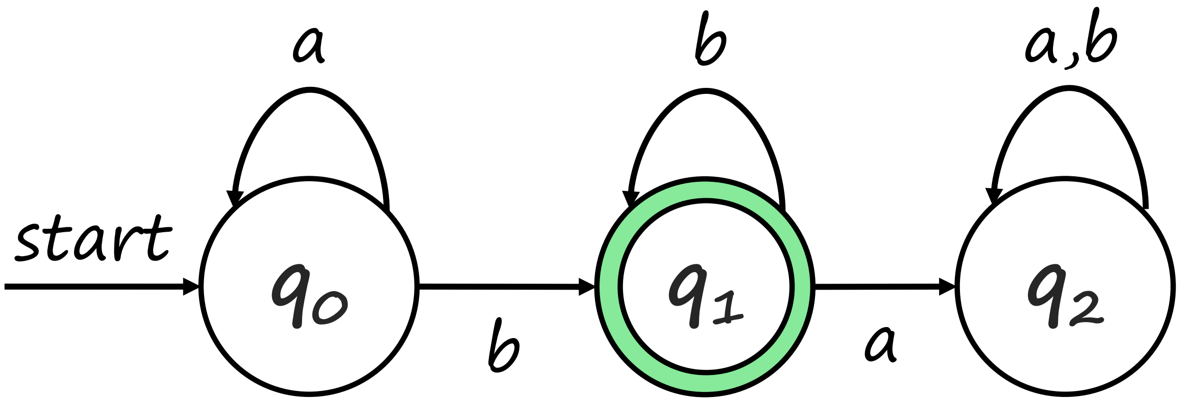
- A **deterministic finite automaton(DFA)** consists of:
  - Q:A finite set of states
  - Σ: A finite set of input symbols
  - δ: A transition function. $Q \times \Sigma \rightarrow Q$
  - q_0: The start state
  - F: A set of final or accepting states. $F \subseteq Q$
- We can donate a DFA as a "five tuple"

$$A = (Q, \Sigma, \delta, q_0, F)$$

Deterministic Finite Automaton(DFA)

- A DFA takes a finite string(sequence of input symbols) as input, and process the input symbols of the string one by one, and change its state according to the transition function δ.
- DFA will always stop, if the state is in the set of accepting states(F) when a DFA stops, we say that this DFA **accepts** the input. Otherwise it **rejects** the input.

Simpler Notations for DFA



Transition Diagram

start state

	a	b
$\rightarrow q_0$	q_0	q_1
$* q_1$	q_2	q_1
q_2	q_2	q_2

final state

Transition Table

Extending the Transition Function

For a DFA $(Q, \Sigma, \delta, q_0, F)$, we can define an ***extended transition function***

$$\hat{\delta}: Q \times \Sigma^* \to Q$$

$\hat{\delta}$ takes a state q and a string w as input, and returns the state after processing string w from state q.

- $\hat{\delta}(q, \epsilon) = q$
- If $w = xa$, where $a \in \Sigma$. Then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

Extending the Transition Function

- $\hat{\delta}(q_0, \epsilon) = q_0$
- $\hat{\delta}(q_0, 1) = \delta\big(\hat{\delta}(q_0, \epsilon), 1\big) = \delta(q_0, 1) = q_1$
- $\hat{\delta}(q_0, 11) = \delta\big(\hat{\delta}(q_0, 1), 1\big) = \delta(q_1, 1) = q_0$
- $\hat{\delta}(q_0, 110) = \delta\big(\hat{\delta}(q_0, 11), 0\big) = \delta(q_0, 0) = q_2$
- $\hat{\delta}(q_0, 1101) = \delta\big(\hat{\delta}(q_0, 110), 1\big) = \delta(q_2, 1) = q_3$
- $\hat{\delta}(q_0, 11010) = \delta\big(\hat{\delta}(q_0, 1101), 0\big) = \delta(q_3, 0) = q_1$
- $\hat{\delta}(q_0, 110101) = \delta\big(\hat{\delta}(q_0, 11010), 1\big) = \delta(q_1, 1) = q_0$

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

The Language of DFA

- For a DFA $\boldsymbol{A} = (Q, \Sigma, \delta, q_0, F)$, the language of $\boldsymbol{A}$, donated as $L(\boldsymbol{A})$, is the set of all string that A accepts.

$$L(A) = \{w | \hat{\delta}(q_0, w) \in F\}$$

- For a certain language L, if there exists a DFA $\boldsymbol{A}$ such that $L = L(\boldsymbol{A})$, then we call L is a **regular language(正则语言)**.

We'll not take much time in DFA and regular language. Feel free to read chapter 2,3,4 in IALC if you are interested in it.

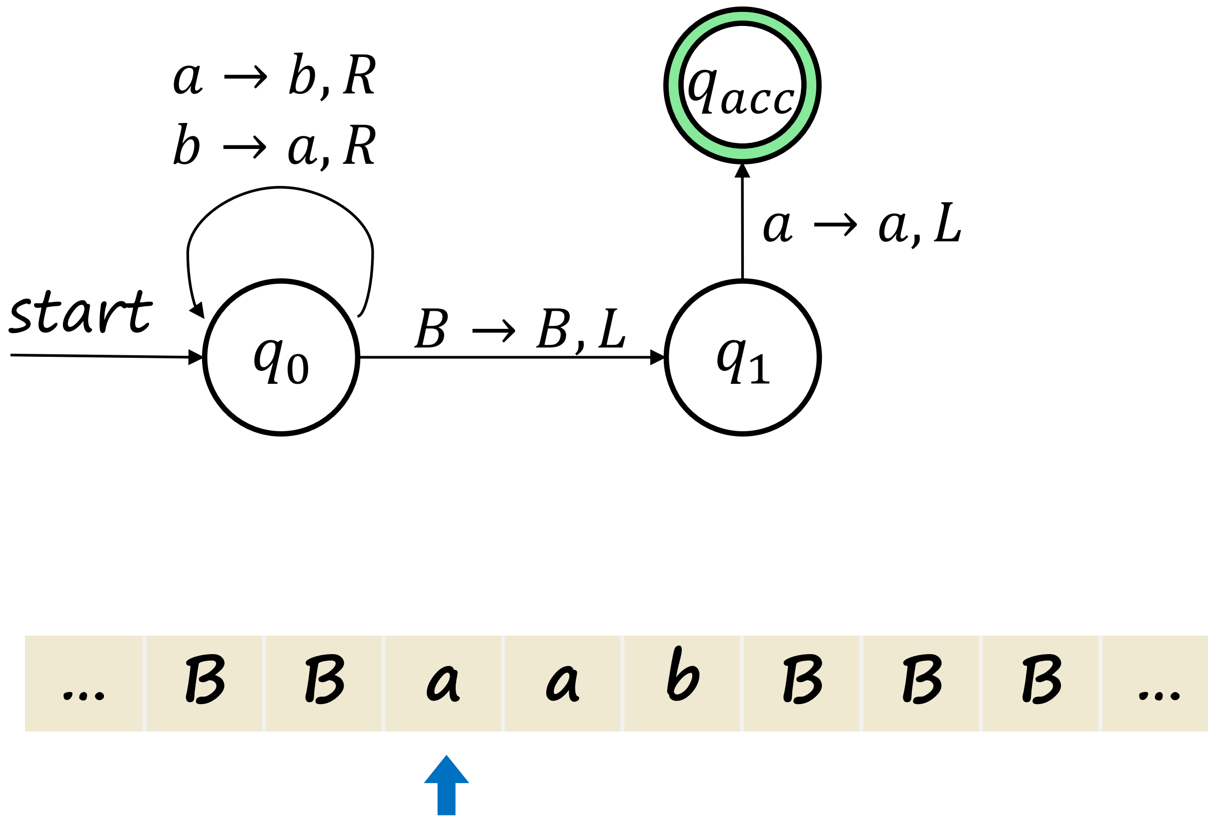
But DFA can not recognize all the languages.

The Limitation of DFA

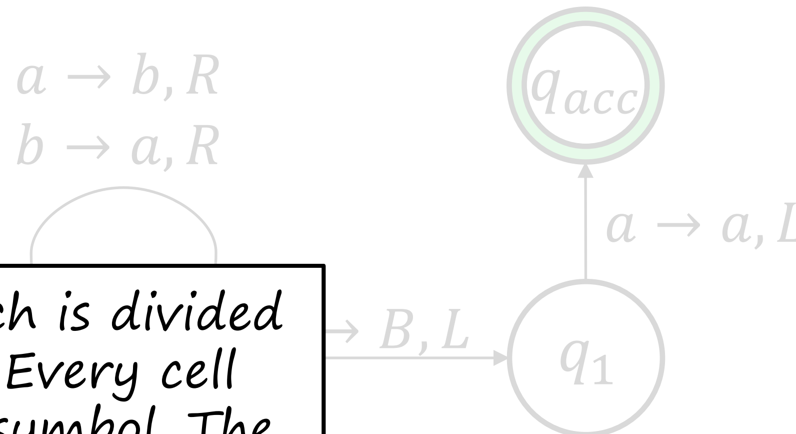
- There is no DFA for languages like $\{0^n1^n | n \in \mathbb{N}^+\}$.
- We need a more powerful "machine" to model our computer.

Turning Machine!!

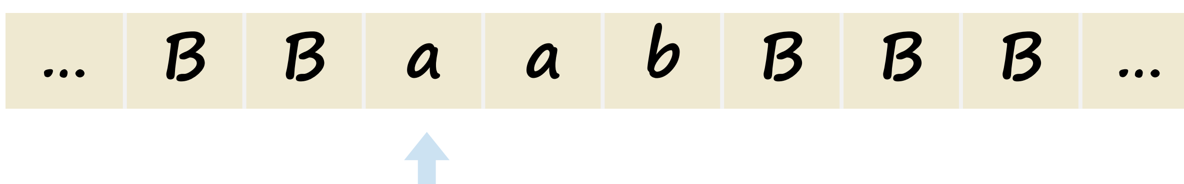
Turing Machine



Turing Machine

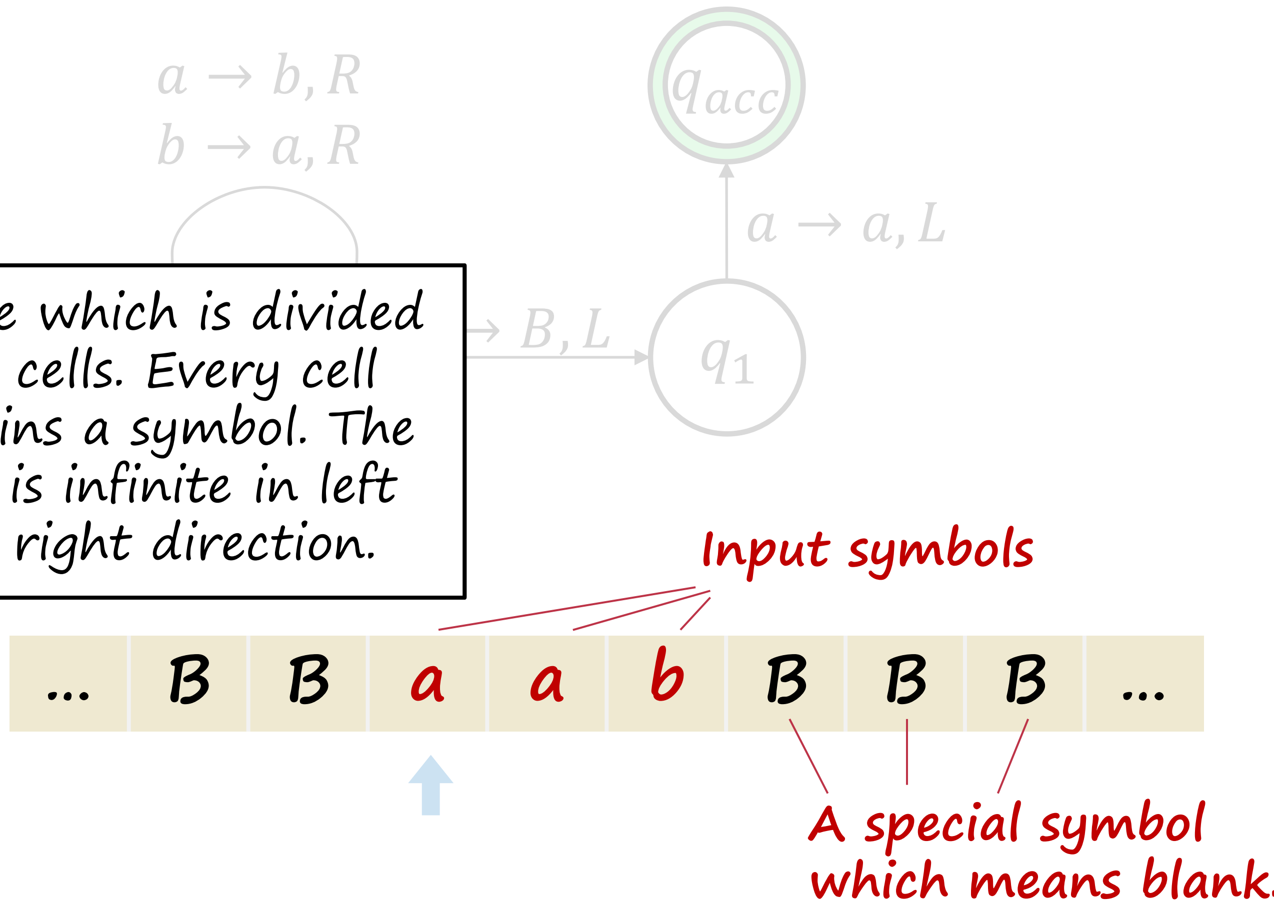


A tape which is divided into cells. Every cell contains a symbol. The tape is infinite in left and right direction.

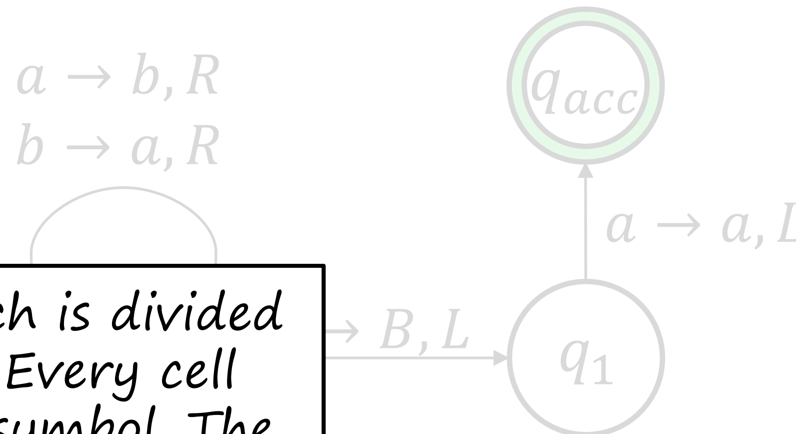


Turing Machine

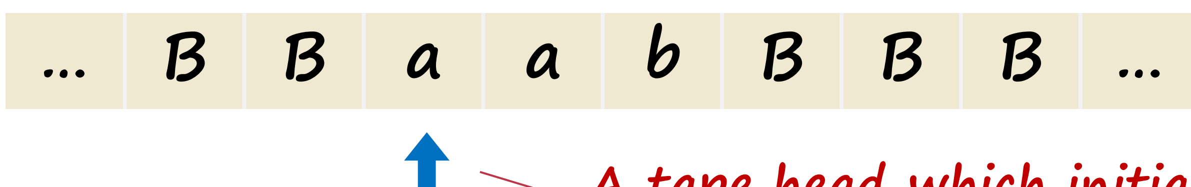
A tape which is divided into cells. Every cell contains a symbol. The tape is infinite in left and right direction.



Turing Machine

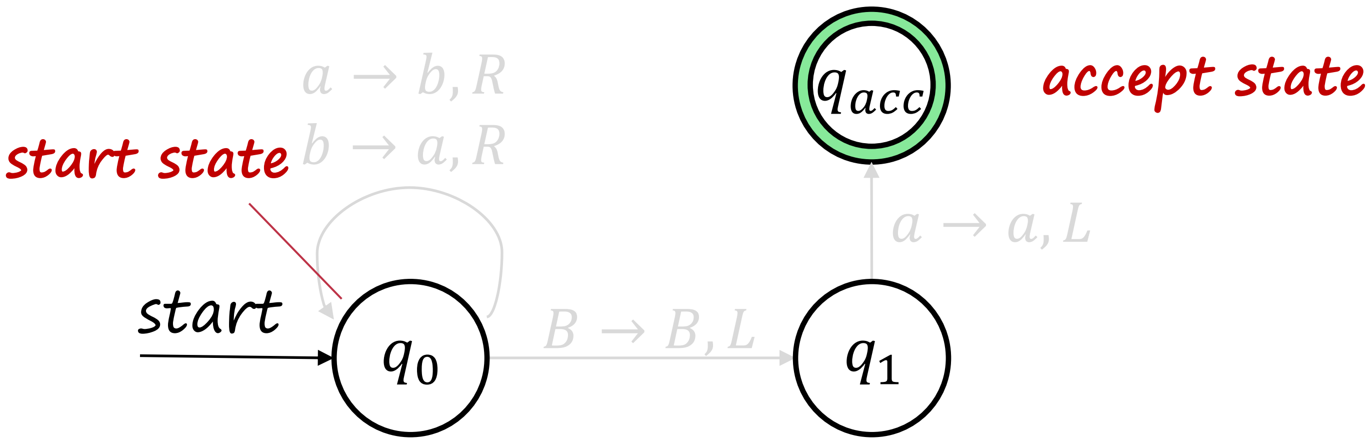


A tape which is divided into cells. Every cell contains a symbol. The tape is infinite in left and right direction.



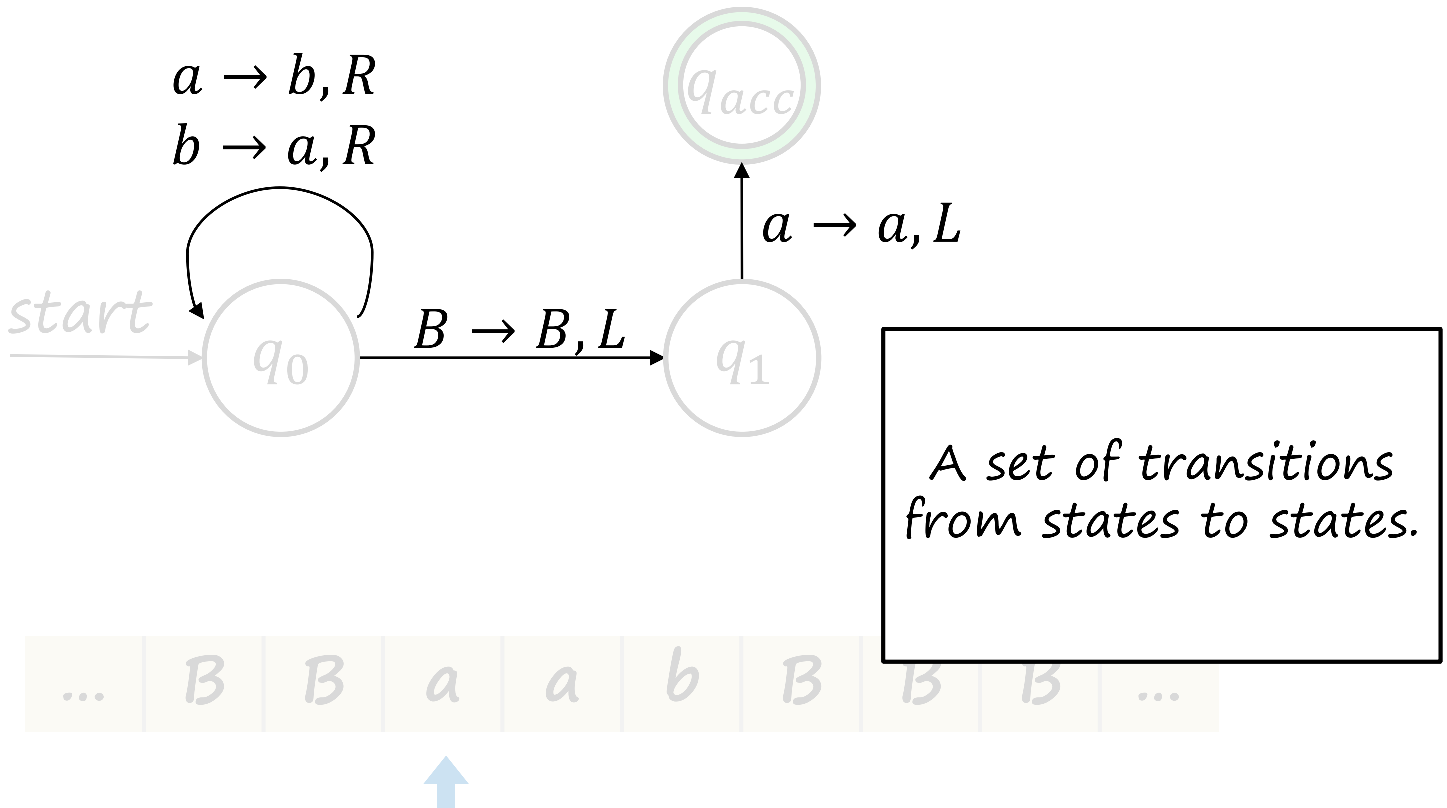
A tape head which initially points to the start of input symbol

Turing Machine

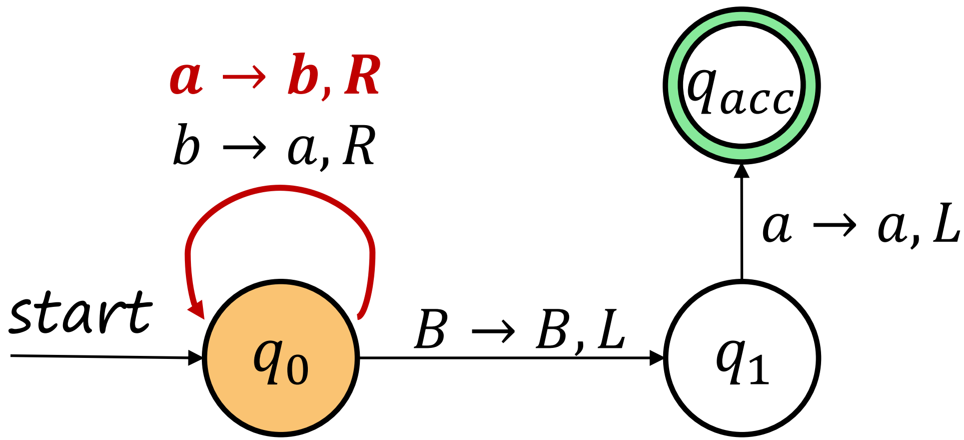


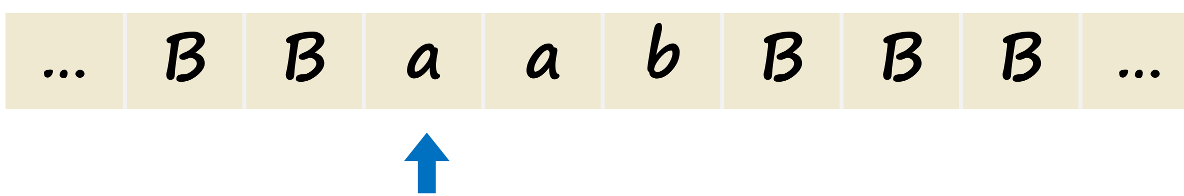
A set of states with a start state and several final or accept state.

Turing Machine

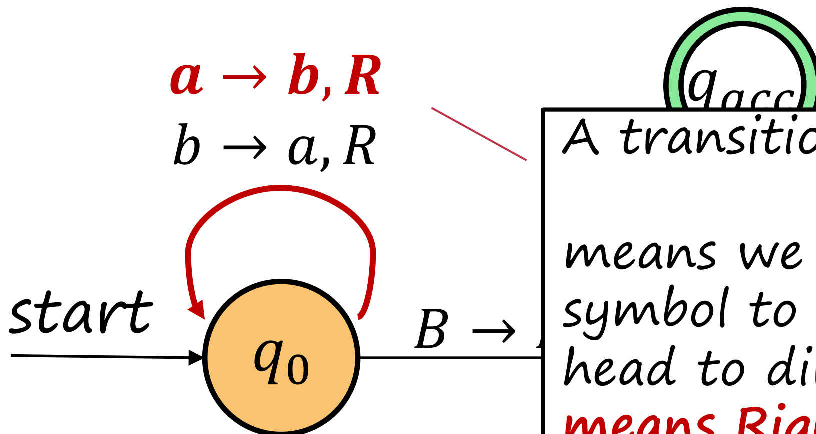


Turing Machine





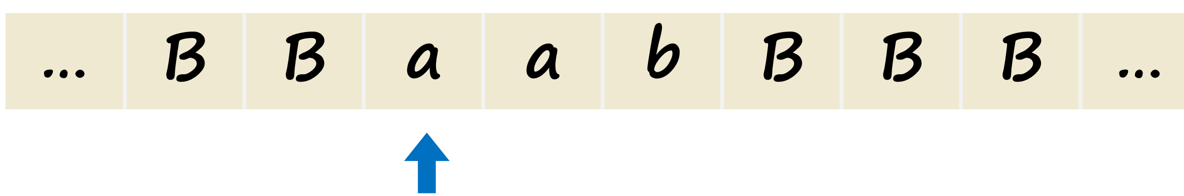
Turing Machine



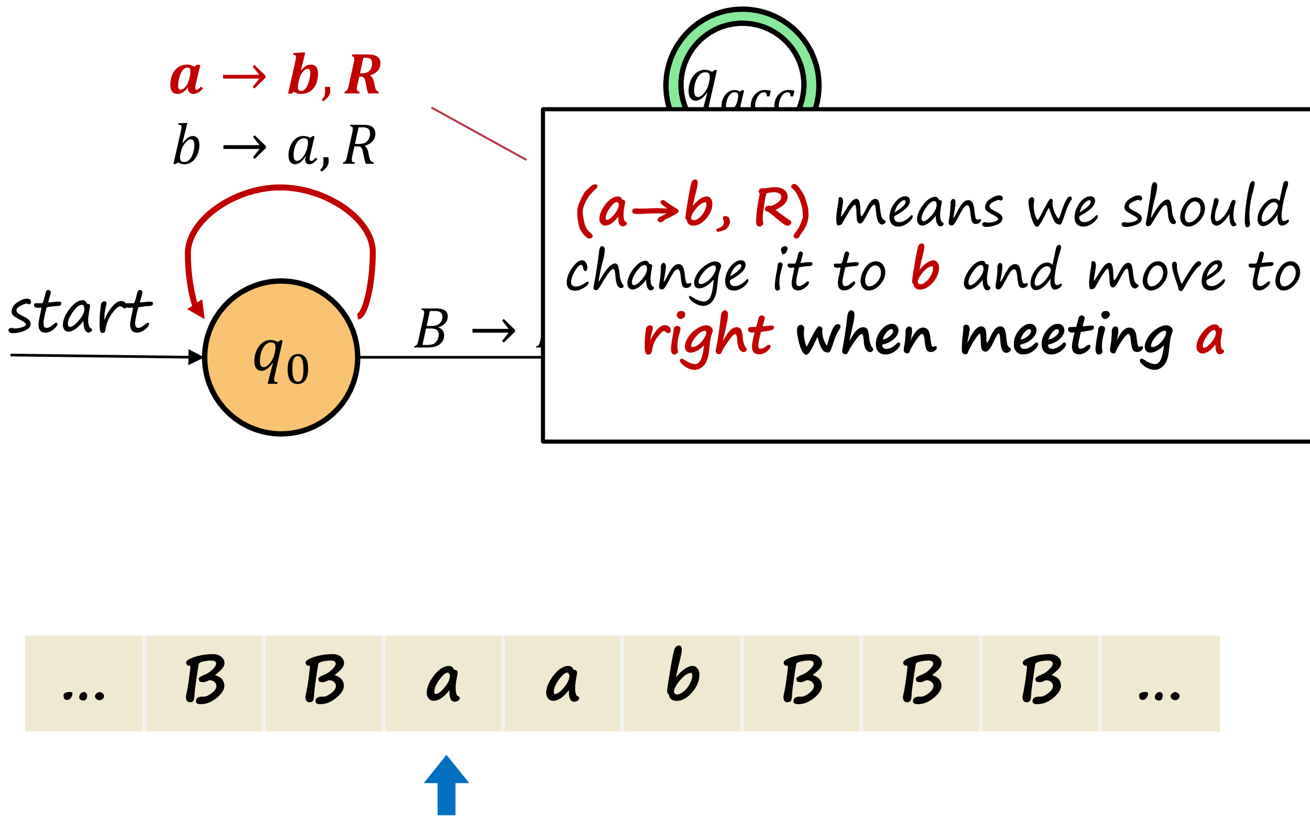
A transition has the form:

(s1→s2, d)

means we should change the symbol to **s2** and move the tape head to direction **d (d∈{R,L}, means Right or Left)** when meeting **s1**.

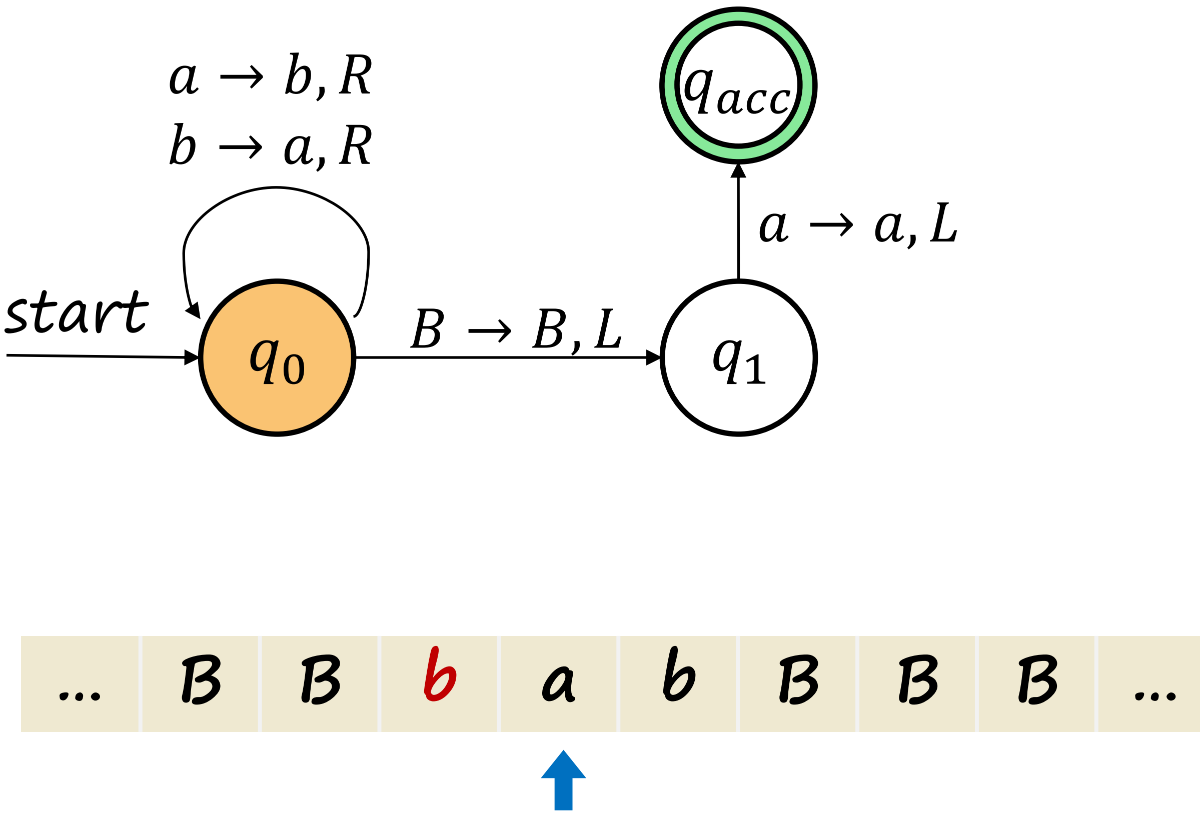


Turing Machine

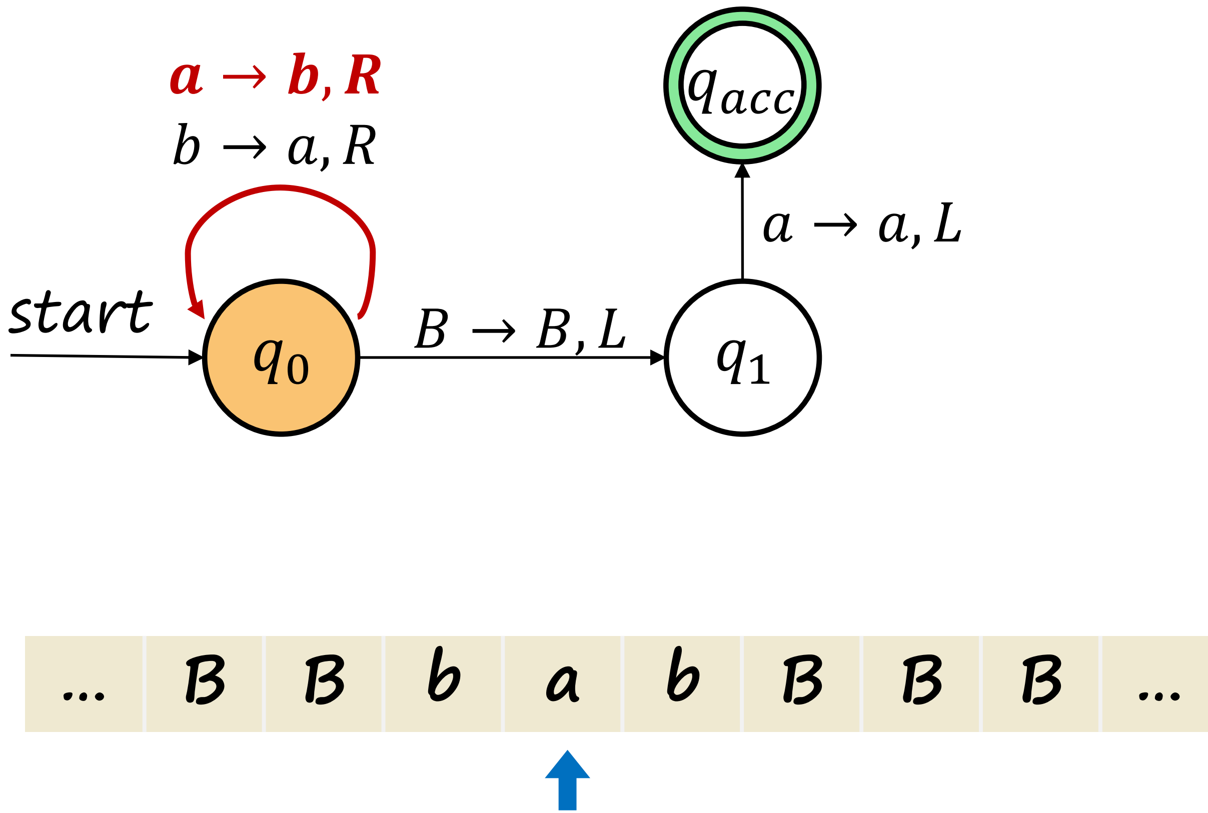


(a→b, R) means we should change it to b and move to right when meeting a

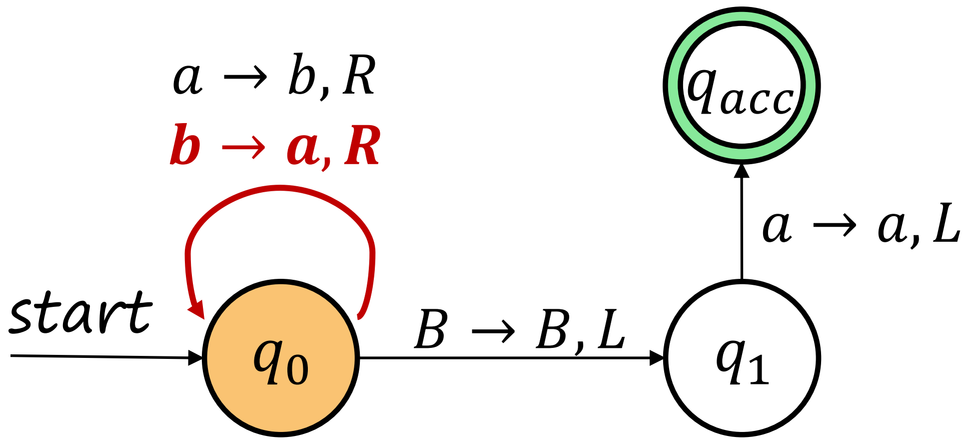
Turing Machine

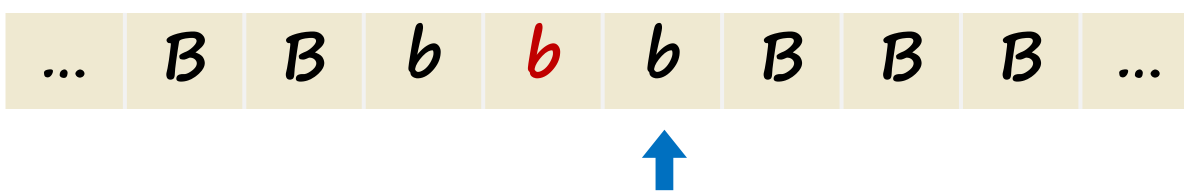


Turing Machine

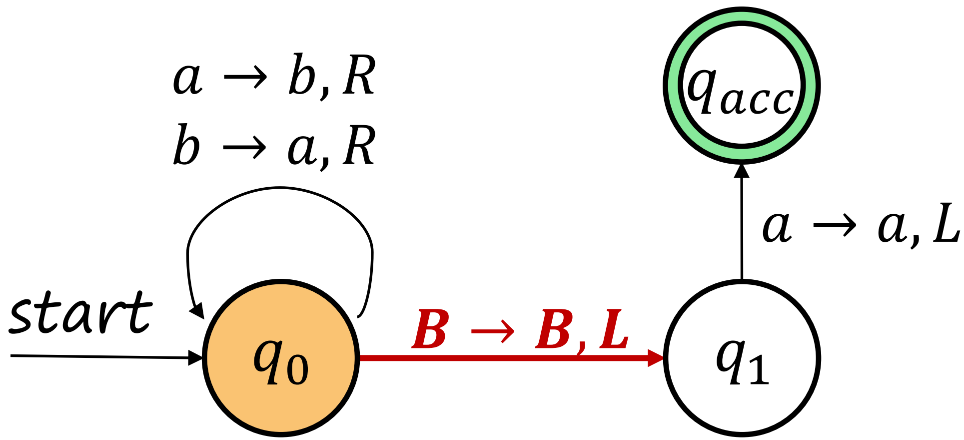


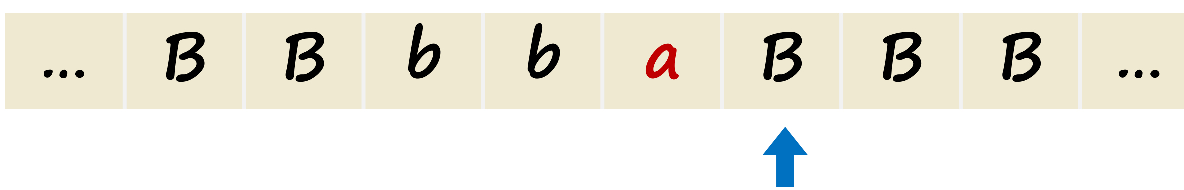
Turing Machine



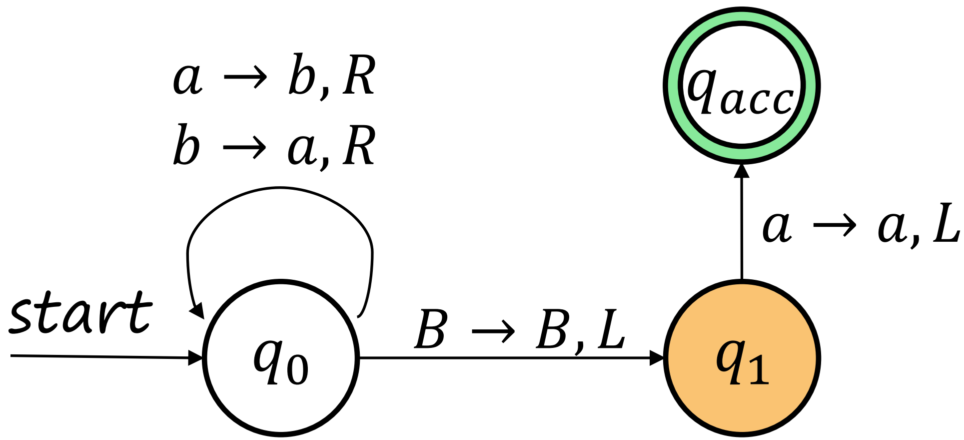


Turing Machine





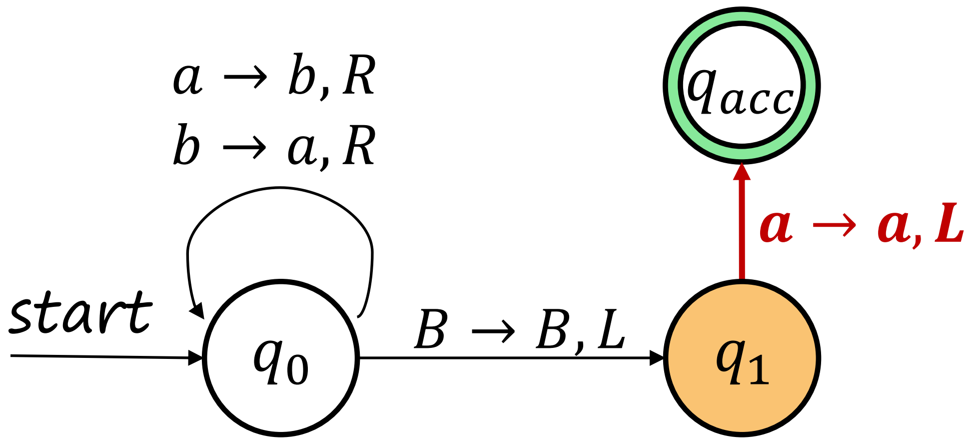
Turing Machine

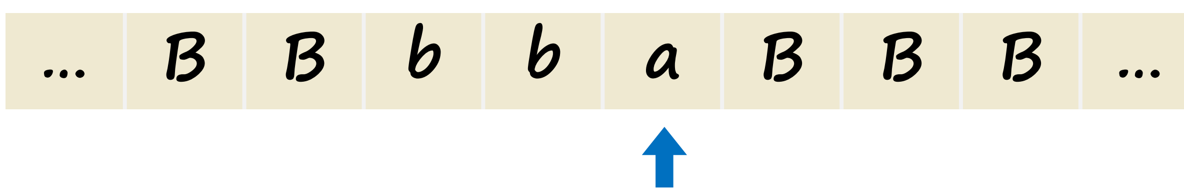


It's ok not to change the tape.

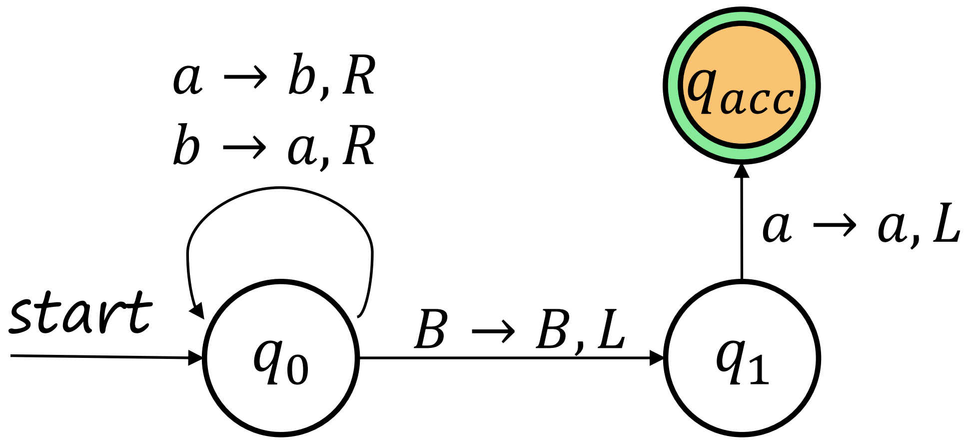
...	B	B	b	b	a	B	B	B	...

Turing Machine





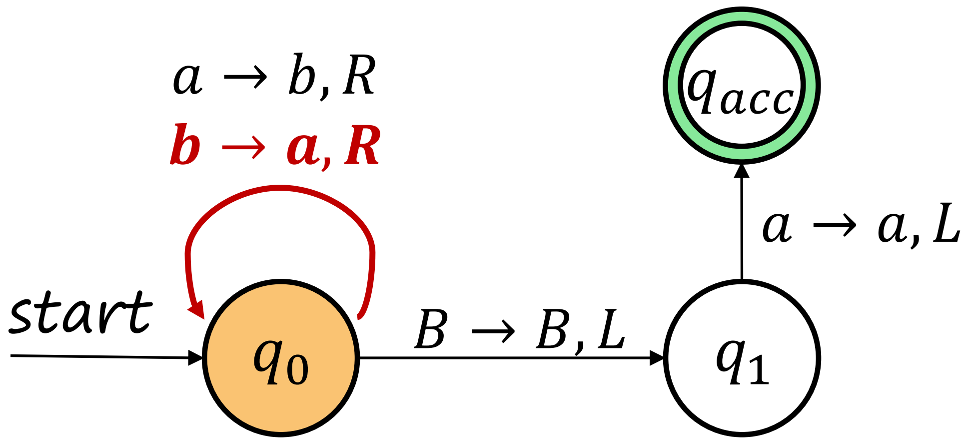
Turing Machine

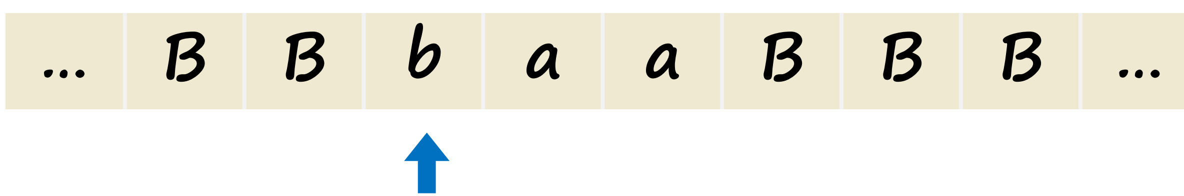


Accept the input **once** we get into the accept state.

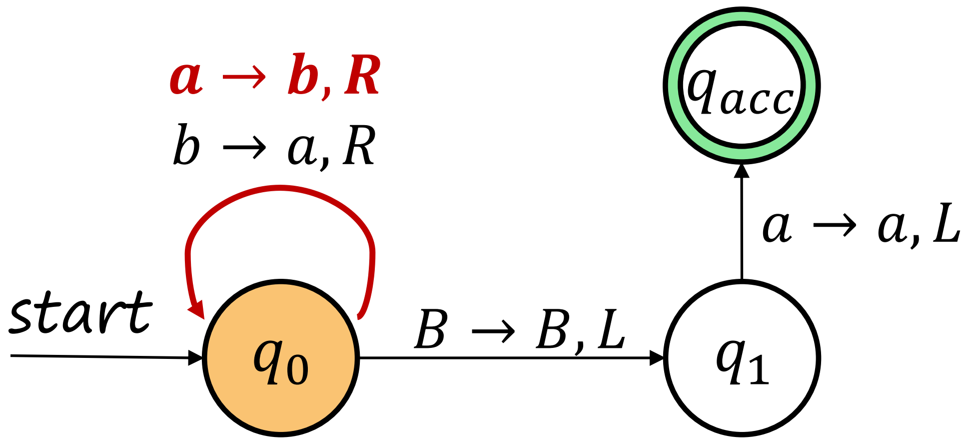
...	B	B	b	b	a	B	B	B	...

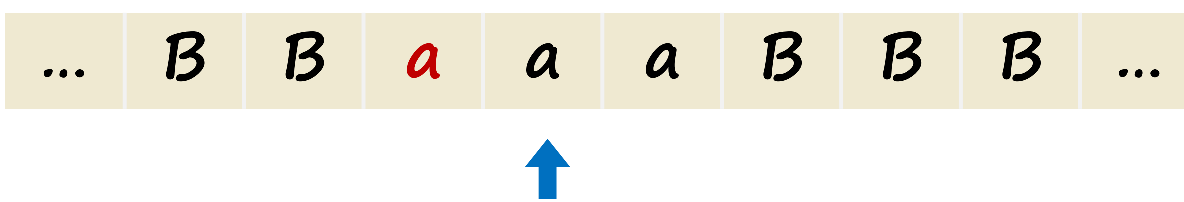
Turing Machine



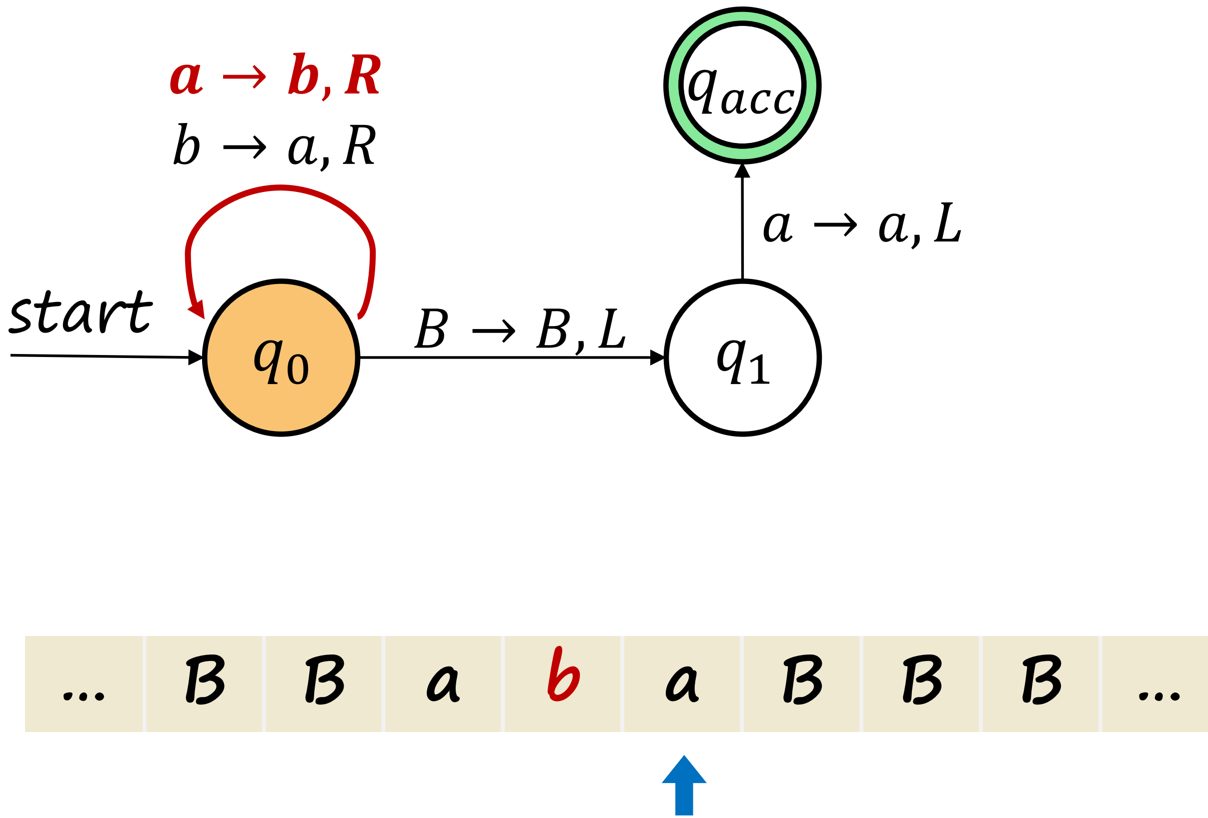


Turing Machine

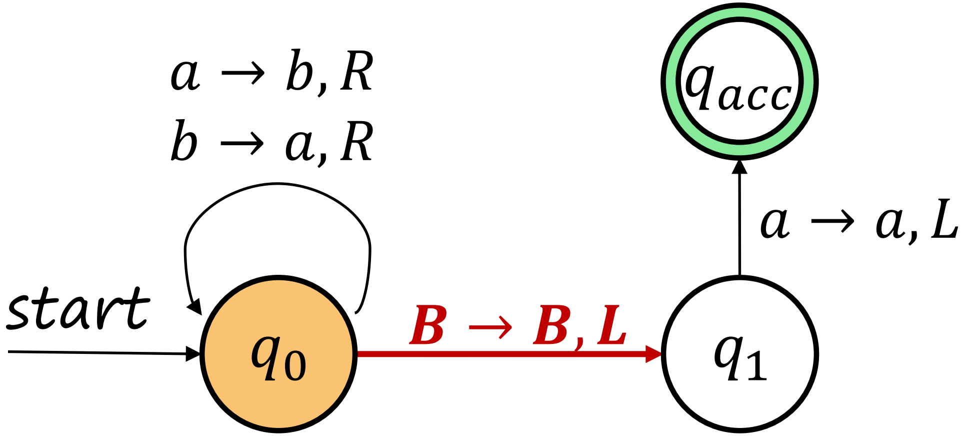


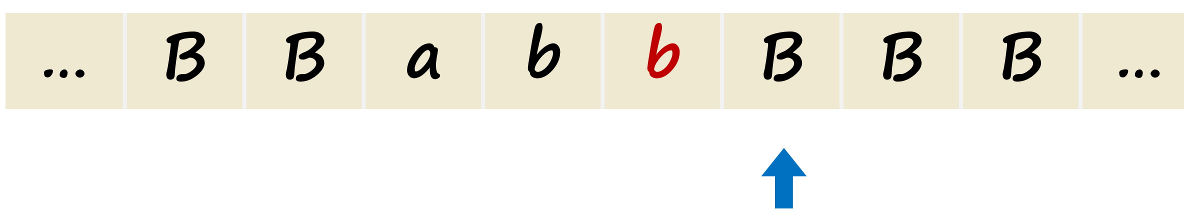


Turing Machine

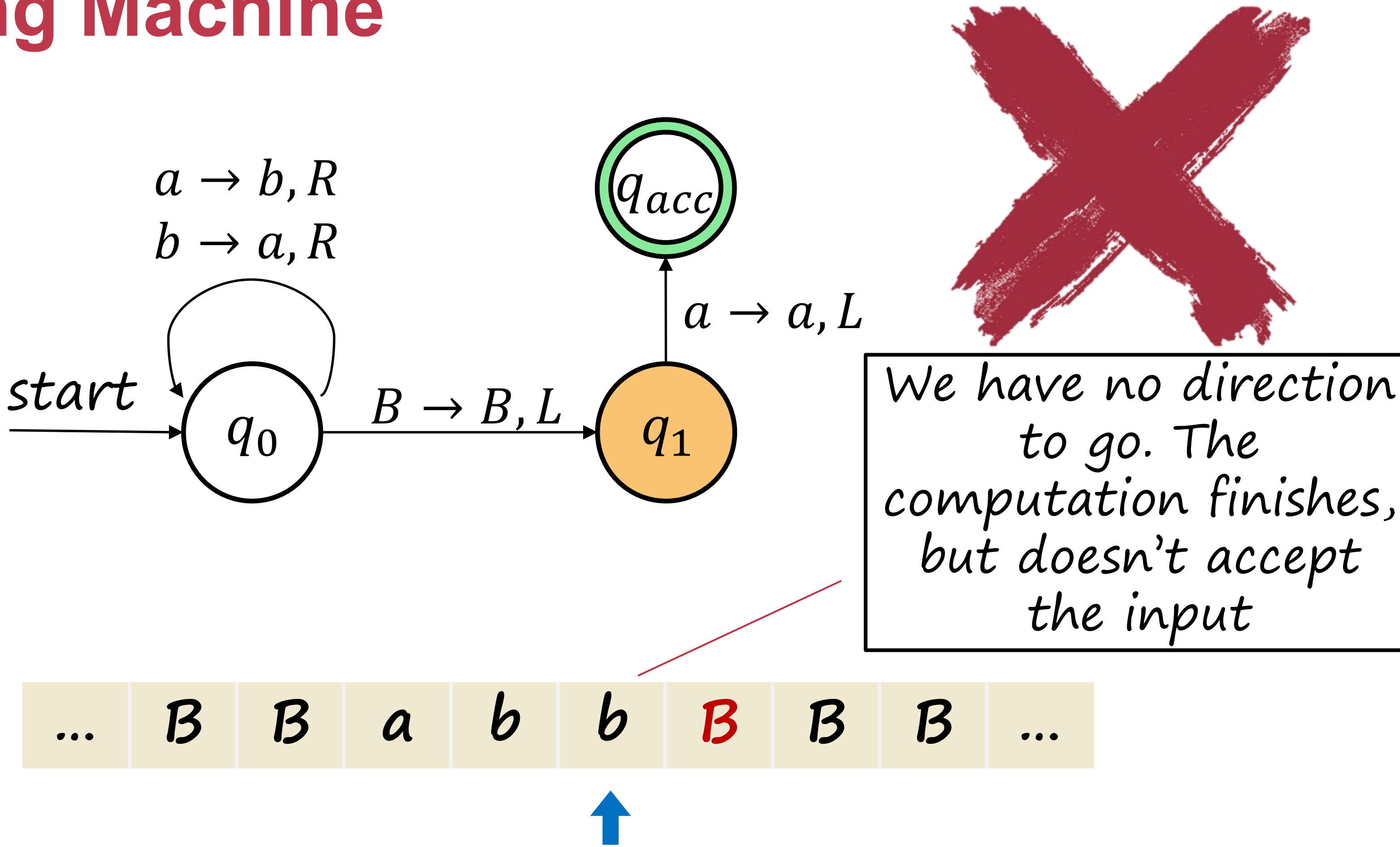


Turing Machine

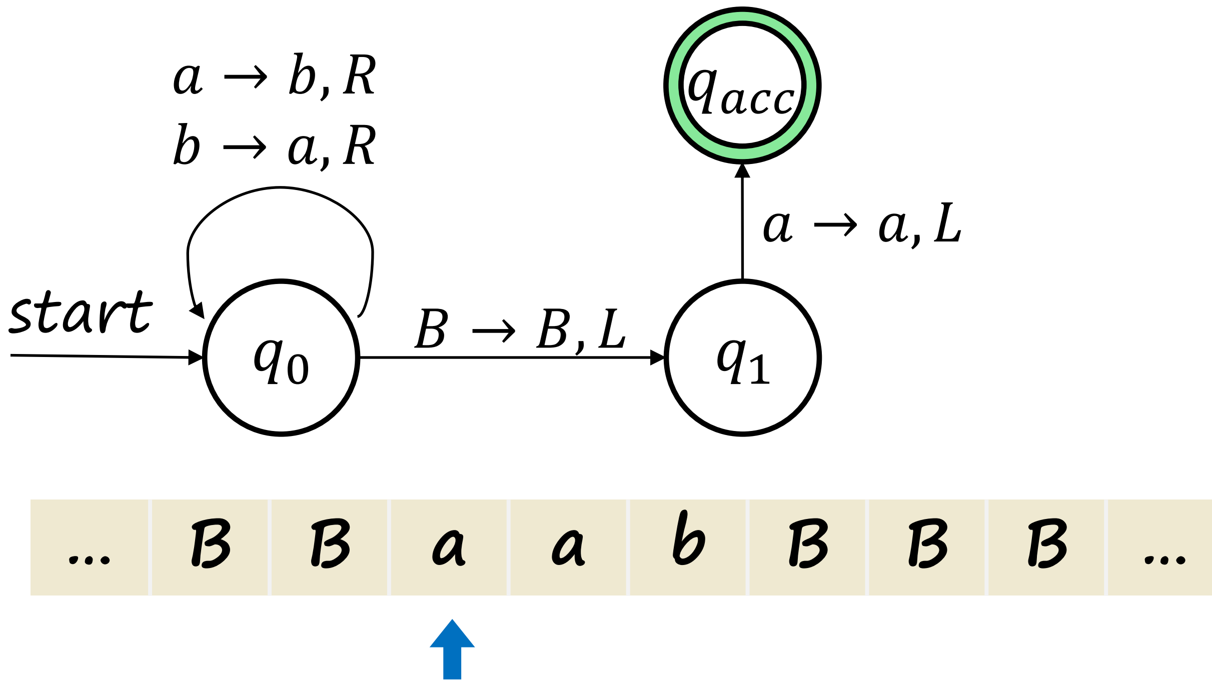




Turing Machine



Turing Machine



$\{w \mid w \in \{a,b\}^* \wedge (w \text{ ends with } b)\}$

Turing Machine

- A **Turing Machine** consists of:
  - Q:A finite set of states
  - Σ: A finite set of input symbols
  - Γ: The complete set of tape symbols. $\Sigma \subset \Gamma$
  - δ: A transition function. $Q \times \Sigma \rightarrow Q \times \Gamma \times \{L, R\}$
  - q_0: The start state. $q_0 \in Q$
  - B: The blank symbol. $B \in \Gamma \wedge B \notin \Sigma$.
  - F: A set of final or accepting states. $F \subseteq Q$
- We can donate a turning machine as a “seven tuple”

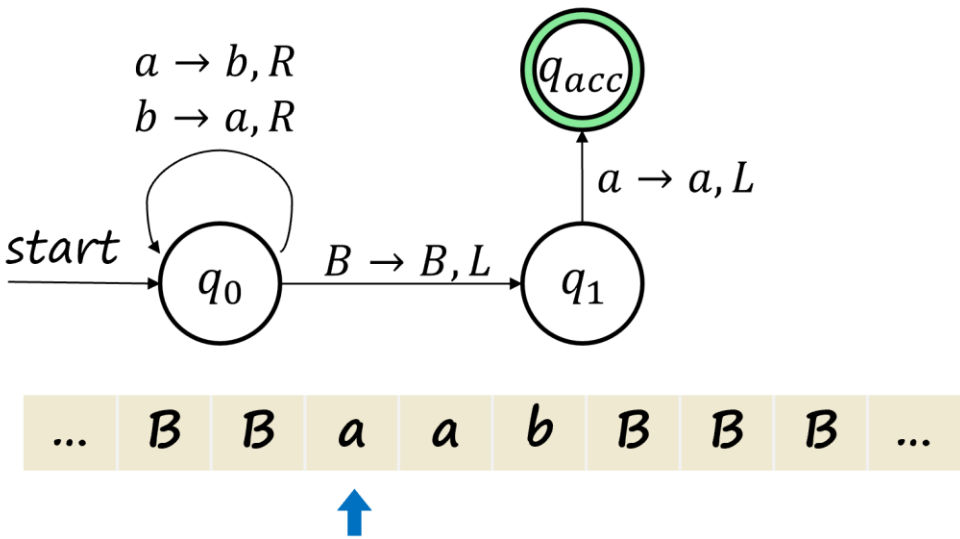
$$A = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

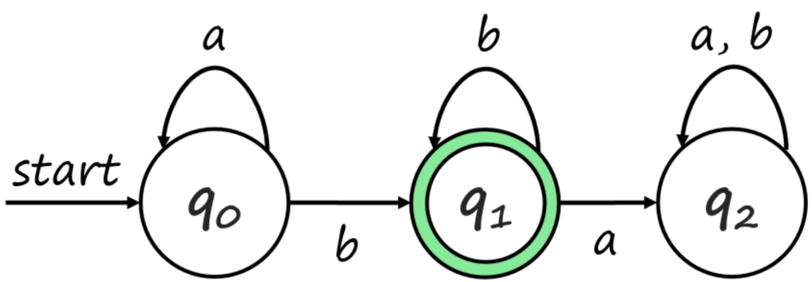
Turing Machine

- Initially the input symbols are put into tape, and the tape heap is pointed to the start of input symbols.
- Turing Machine finishes computation by moving the tape head. In a move, the Turing Machine will:
  - Change state
  - Write a tape symbol in the cell pointed
  - Move the tape head left or right.
- Turing Machine will accept the input once it gets into accept states, or finishes computation with rejection when no move can be made.

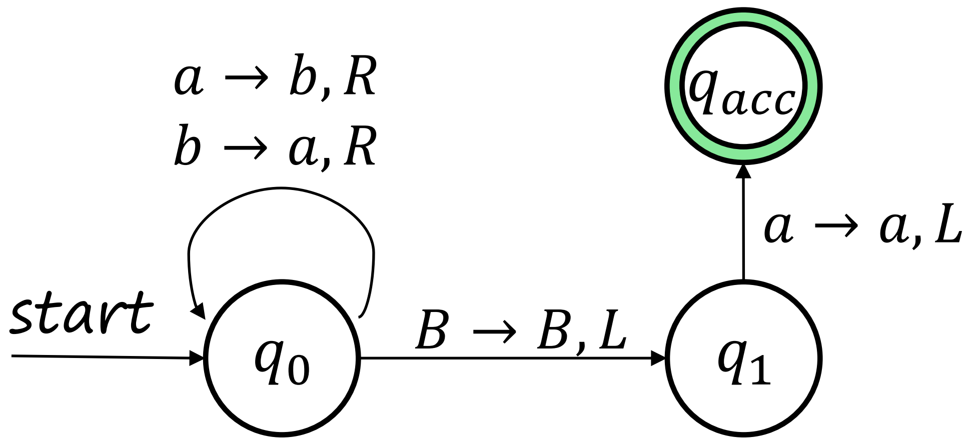
TM and DFA

- DFA has only a bunch of states but TM has an extra "memory-like" tape to store the execution state.
- TM has more capability than DFA
  - A DFA can be converted into a TM





Simpler Notations for TM



Transition Diagram

start state

final state

	a	b	B
$\to q_0$	(q_0, b, R)	(q_0, a, R)	(q_1, B, L)
q_1	(q_{acc}, a, L)	$-$	$-$
$* \; q_{acc}$	$-$	$-$	$-$

Transition Table

The Language of a TM

- Similar to DFA, the language of a TM $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\Gamma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ is set of strings that it accepts, or:

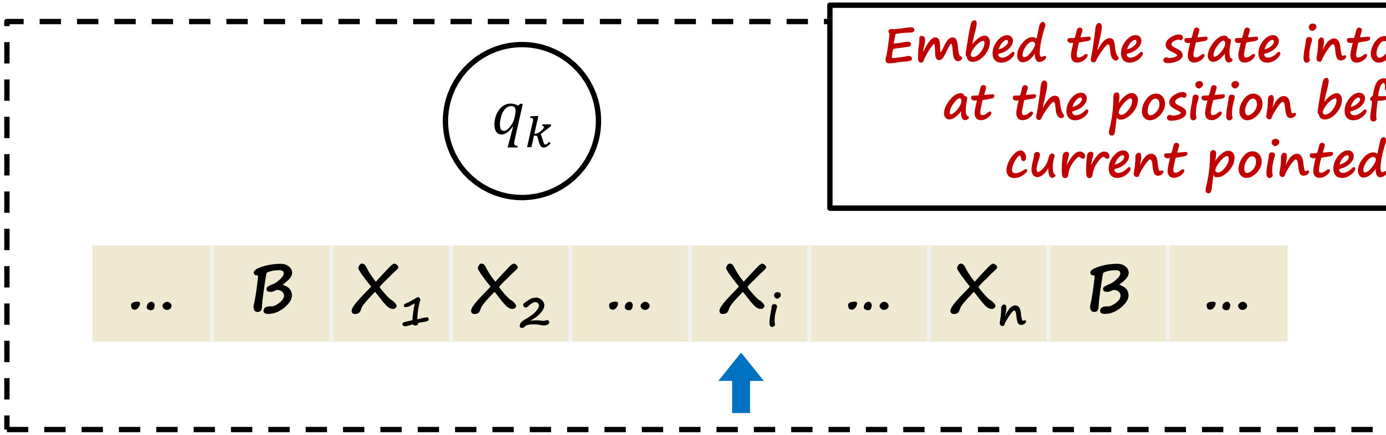
$$L(\boldsymbol{M}) = \{w \mid w \in \Sigma^* \wedge p \in F \wedge q_0 w \vdash^* \alpha p \beta\}$$

- For a certain language L, if there exists a TM $\boldsymbol{M}$ such that $L = L(\boldsymbol{M})$, then we call L is a **recursively enumerable language(递归可枚举语言)**， or ***RE***.

We'll talk more about RE in the next courses.

Instantaneous Descriptions for TM

- **Instantaneous Descriptions (ID's)** formally describe what a Turing machine does.
  - Tape contents, Current state, Location of type head



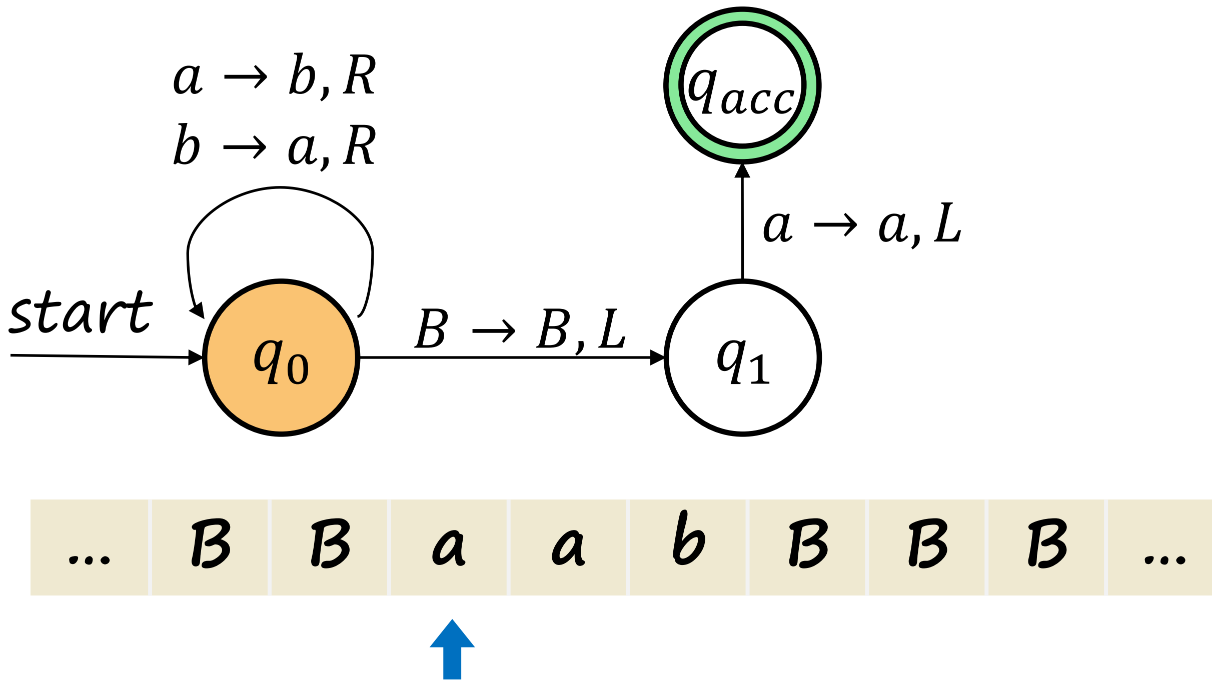
Embed the state into the tape at the position before the current pointed cell.

$$X_1X_2 \ldots X_{i-1}q_kX_i \ldots X_n$$

Instantaneous Descriptions(ID's) for TM

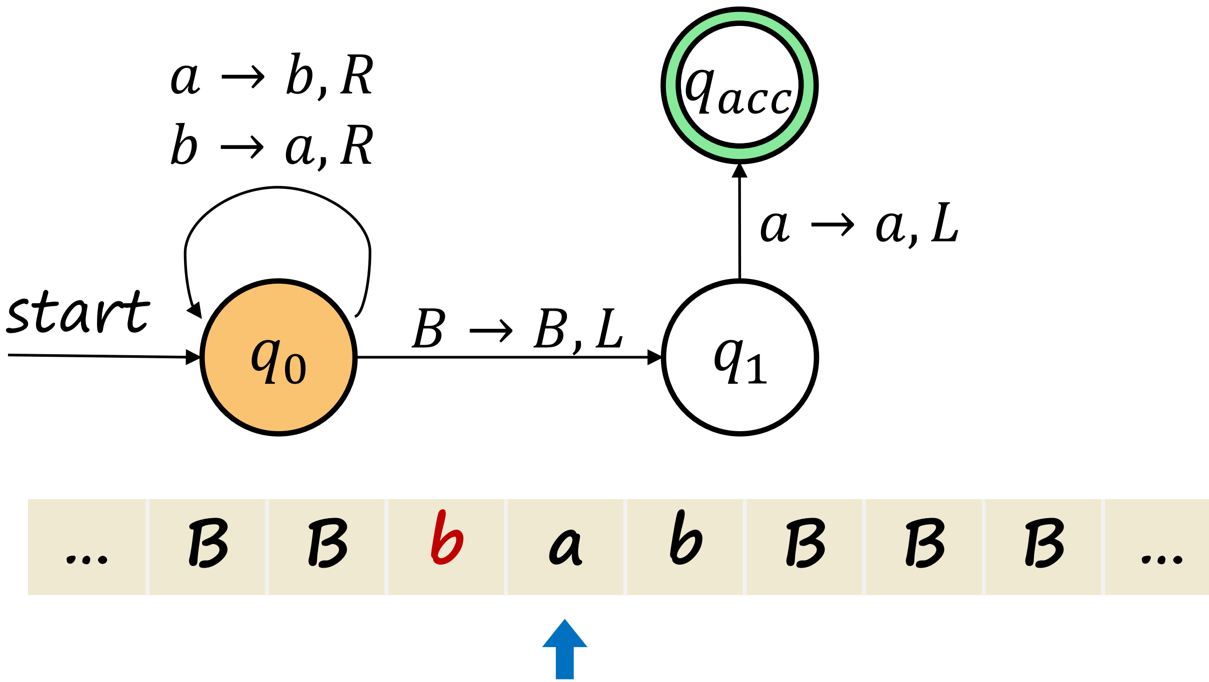
- Describe the moves of TM as $\vdash_M$
- If $\delta(q, X_i) = (p, Y, R)$ then
  - $X_1X_2 \dots X_{i-1}\boldsymbol{qX_i}X_{i+1} \dots X_n \ \vdash_M \ X_1X_2 \dots X_{i-1}\boldsymbol{Yp}X_{i+1} \dots X_n$
- If $\delta(q, X_i) = (p, Y, L)$ then
  - $X_1X_2 \dots X_{i-1}\boldsymbol{qX_i}X_{i+1} \dots X_n \ \vdash_M \ X_1X_2 \dots \boldsymbol{p}X_{i-1}\boldsymbol{Y}X_{i+1} \dots X_n$
- $\vdash_M^*$ indicated zero, one or more moves
  - $ID_1 \vdash_M^* ID_2$ means that ID_1 can be transformed to ID_2 through zero, one or more moves.

ID's and Moves



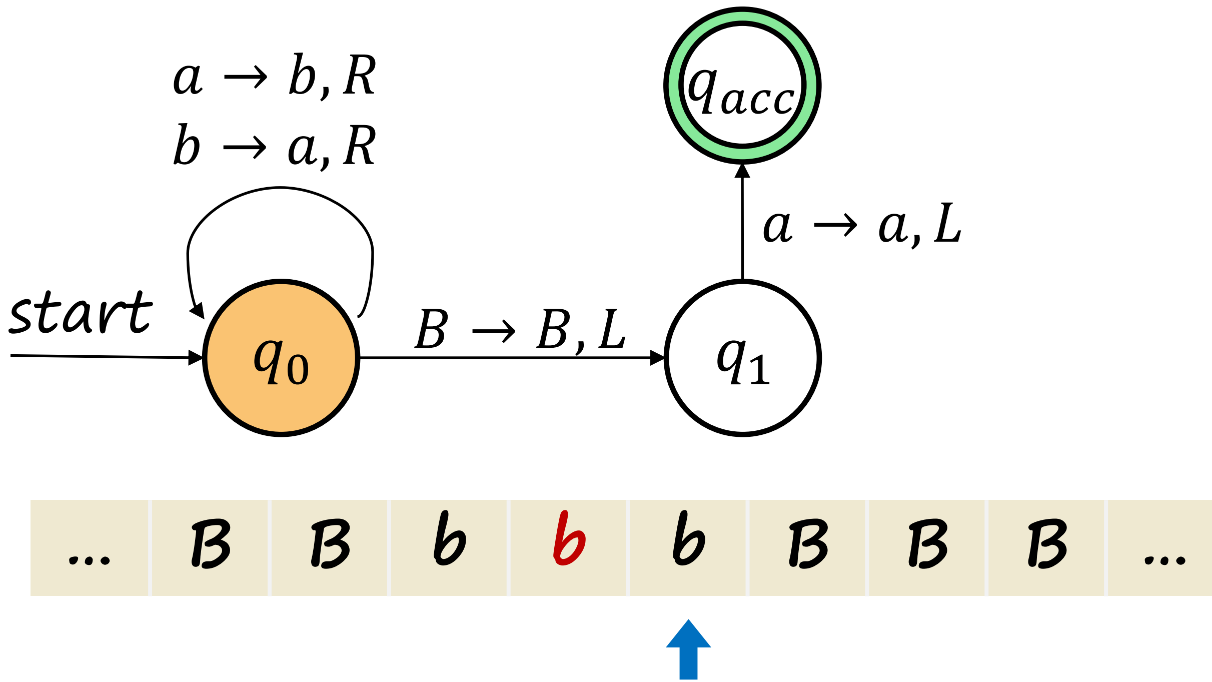
q_0aab

ID's and Moves



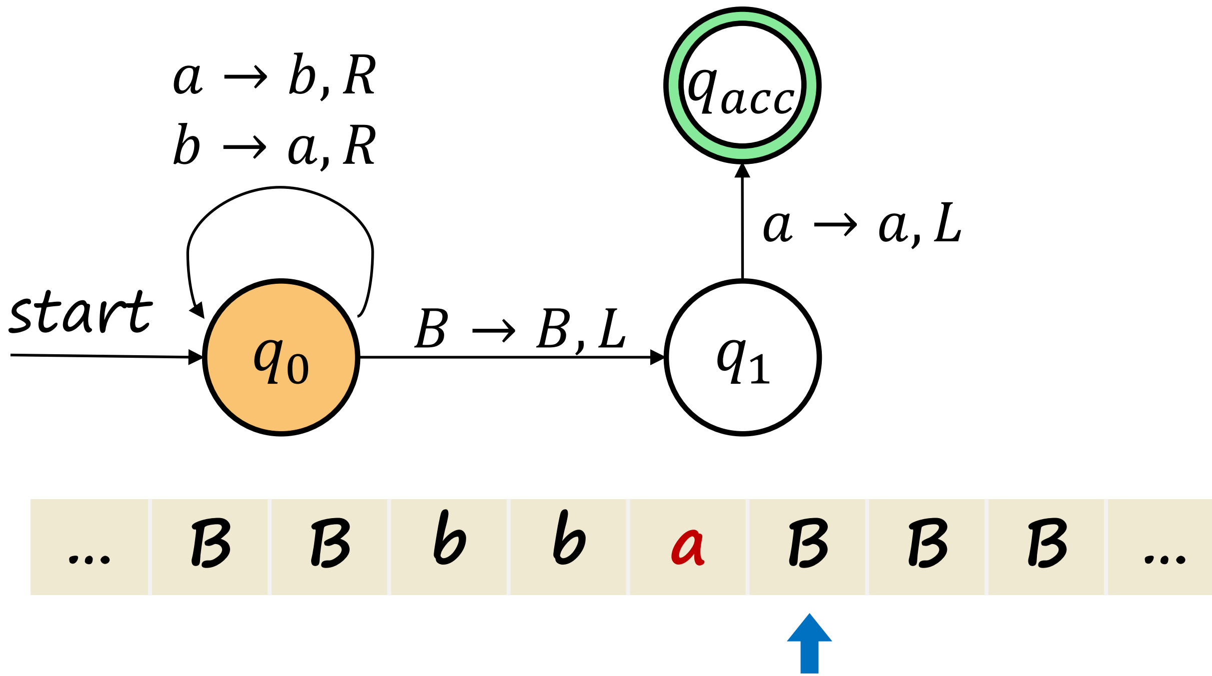
$q_0aab \vdash_M bq_0ab$

ID's and Moves



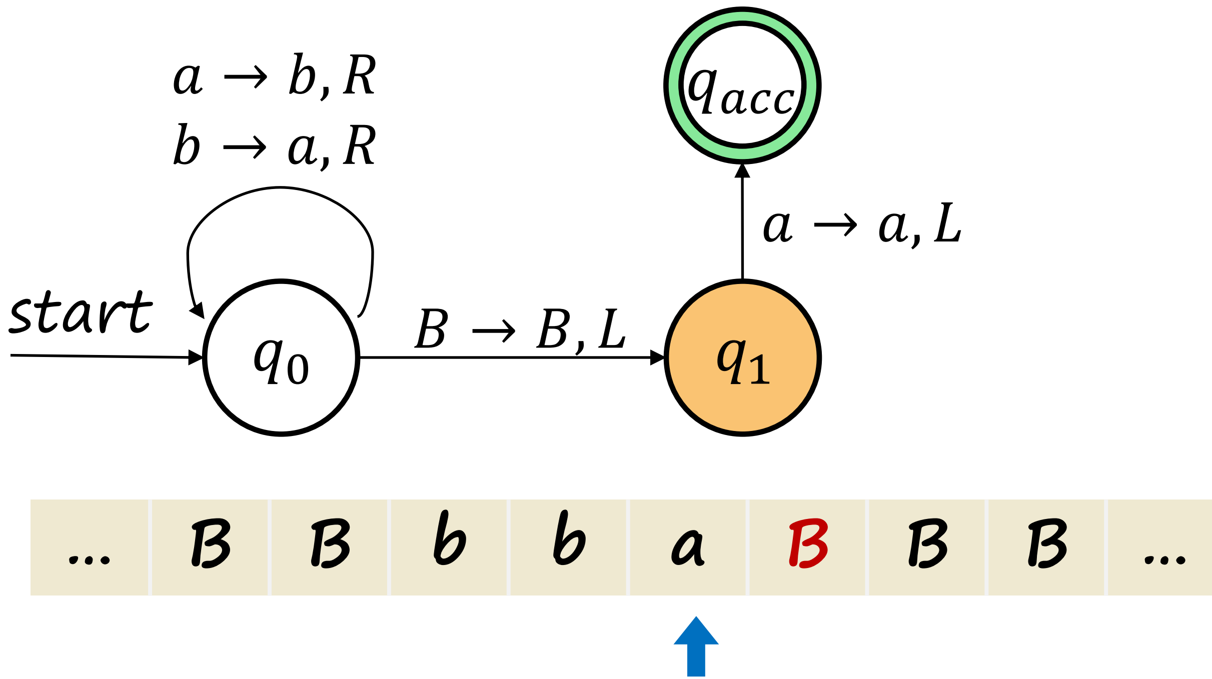
$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b$

ID's and Moves



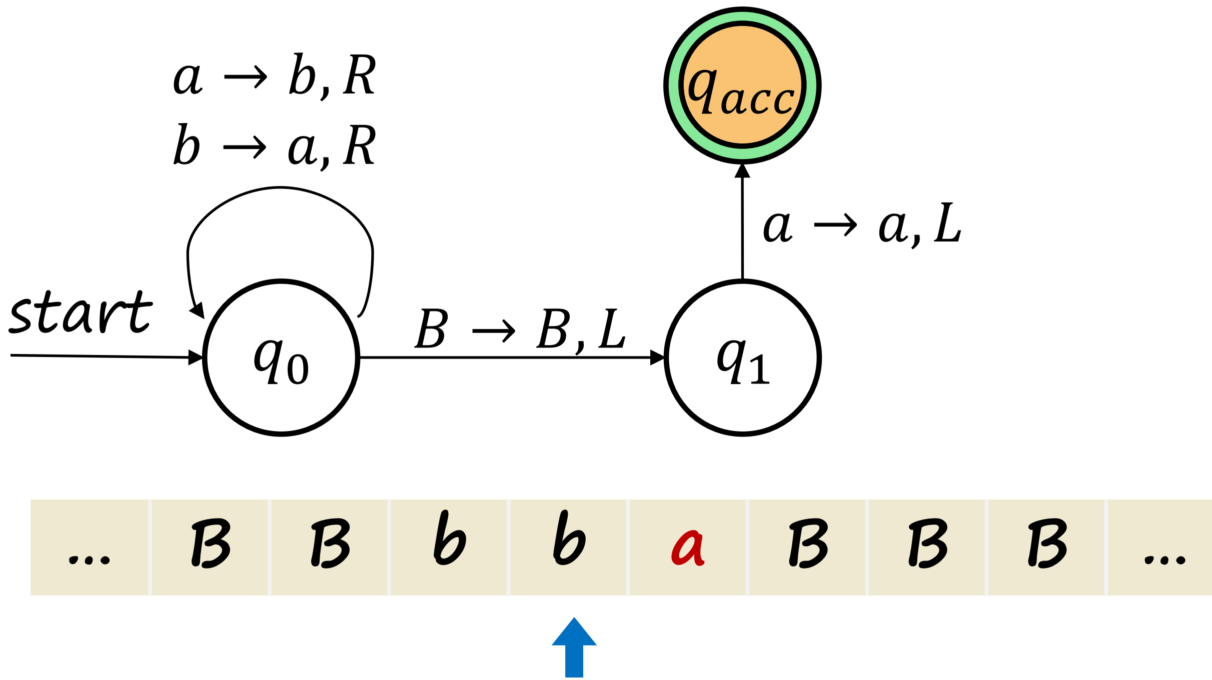
$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M$
$bbaq_0B$

ID's and Moves



$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M$
$bbaq_0B \vdash_M bbq_1aB$

ID's and Moves



$$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M$$
$$bbaq_0B \vdash_M bbq_1aB \vdash_M bq_{acc}ba$$

Example: TM for $\{0^n 1^n\}$

- Count the number of 0 and the number of 1, and compare them.
  - But we have no idea how to store the number and compare them.
- Or when we see a 0, we try to find an 1, and repeat the process until we count every symbol.

Example: TM for $\{\mathbf{0}^n\mathbf{1}^n\}$

- Our solution:
  - When we see a $\mathbf{0}$ at the beginning, we change this $\mathbf{0}$ to $\boldsymbol{X}$ to indicate we have counted it, and try to find an $\mathbf{1}$ by moving tape head to right.
  - When we find an $\mathbf{1}$, we change this $\mathbf{1}$ to $\boldsymbol{Y}$, and turn back to find another $\mathbf{0}$ at the beginning (right after an $\boldsymbol{X}$).
  - Repeat the above process when there are only $\boldsymbol{X}$ and $\boldsymbol{Y}$ left on the tape, then we accept the input. Otherwise we reject the input.

Example: TM for $\{0^n 1^n\}$

We've found a 0

...	B	0	0	0	1	1	1	B	...

Example: TM for $\{0^n1^n\}$

Change it to X

...	B	X	0	0	1	1	1	B	...

Example: TM for $\{0^n 1^n\}$

Move right to find an 1

...	B	X	0	0	1	1	1	B	...

Example: TM for $\{0^n1^n\}$

We've found an 1

...	B	X	0	0	1	1	1	B	...

Example: TM for $\{0^n 1^n\}$

Change it to Y

…	B	X	0	0	Y	1	1	B	…

Example: TM for $\{0^n1^n\}$

Turn back to find 0 at the beginning

...	B	X	0	0	Y	1	1	B	...

Example: TM for $\{0^n1^n\}$

Stop move left when we meet X.

…	B	X	0	0	Y	1	1	B	…

Example: TM for $\{0^n 1^n\}$

Take a right move. We found another 0.

…	B	X	0	0	Y	1	1	B	…

Example: TM for $\{0^n 1^n\}$

Change it to X and continue.

...	B	X	X	0	Y	1	1	B	...

Example: TM for $\{0^n 1^n\}$

Another 1.

…	B	X	X	0	Y	1	1	B	…

Example: TM for $\{0^n 1^n\}$

Change it to Y and turn back again.

...	B	X	X	0	Y	Y	1	B	...

Example: TM for $\{0^n 1^n\}$

Meet X, stop moving left.

...	B	X	X	0	Y	Y	1	B	...

Example: TM for $\{0^n 1^n\}$

Another 0.

...	B	X	X	0	Y	Y	1	B	...

Example: TM for $\{0^n 1^n\}$

Change it to X.

…	B	X	X	X	Y	Y	1	B	…

Example: TM for $\{0^n 1^n\}$

Another 1.

...	B	X	X	X	Y	Y	1	B	...

Example: TM for $\{0^n1^n\}$

Change it and turn back again.

…	B	X	X	X	Y	Y	Y	B	…

Example: TM for $\{0^n1^n\}$

Meet X, take a right move.

...	B	X	X	X	Y	Y	Y	B	...

Example: TM for $\{0^n 1^n\}$

There's no 0 left. We need to check if there's only X and Y left on the tape.

...	B	X	X	X	Y	Y	Y	B	...

Example: TM for $\{0^n 1^n\}$

We only need to check towards the right direction as we know the left are all X.

…	B	X	X	X	Y	Y	Y	B	…

Example: TM for $\{0^n1^n\}$

We stop and accept the result until we meet a B.

...	B	X	X	X	Y	Y	Y	B	...

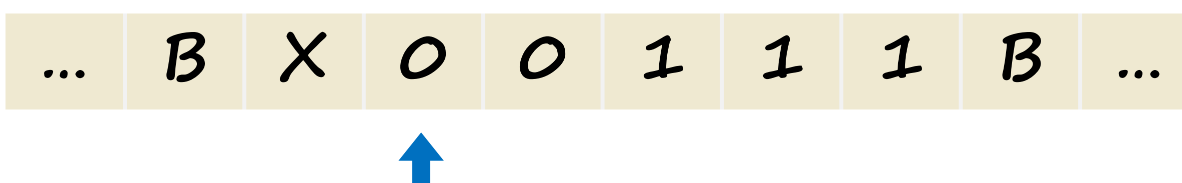
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$					

…	B	0	0	0	1	1	1	B	…

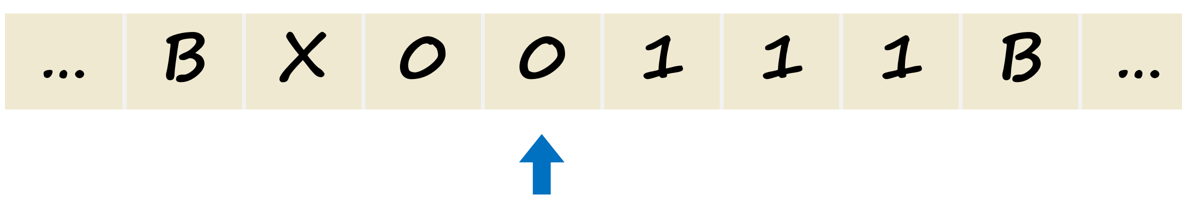
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1					



Example: TM for $\{0^n1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$				



Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$				

…	B	X	0	0	1	1	1	B	…

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)			
q_2					

...	B	X	0	0	Y	1	1	B	...

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)			
q_2	$(q_2, 0, L)$				

…	B	X	0	0	Y	1	1	B	…

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)			
q_2	$(q_2, 0, L)$		$(?, X, R)$		

...	B	X	0	0	Y	1	1	B	...

Example: TM for $\{0^n 1^n\}$

	0	**1**	**X**	**Y**	**B**
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)			
q_2	$(q_2, 0, L)$		(q_0, X, R)		

...	B	X	0	0	Y	1	1	B	...

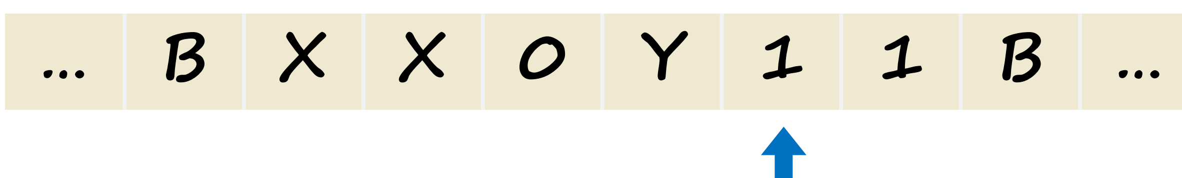
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)			
q_2	$(q_2, 0, L)$		(q_0, X, R)		

...	B	X	X	0	Y	1	1	B	...

Example: TM for $\{0^n1^n\}$

	0	**1**	**X**	**Y**	**B**
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)		



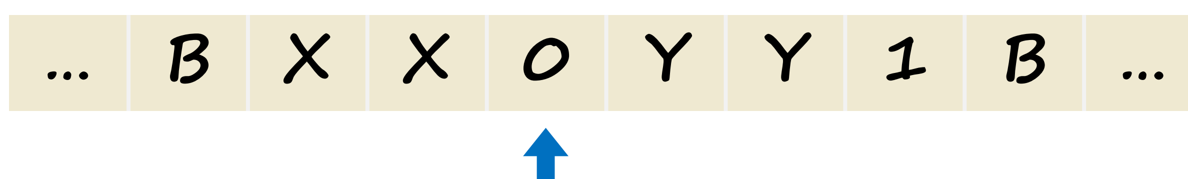
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)		

...	B	X	X	0	Y	Y	1	B	...

Example: TM for $\{0^n 1^n\}$

	0	**1**	**X**	**Y**	**B**
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	



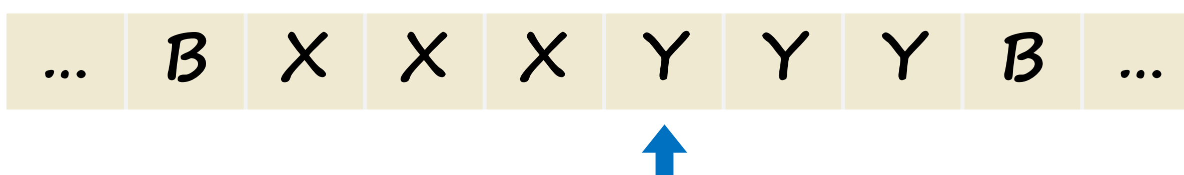
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	

...	B	X	X	X	Y	Y	1	B	...

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)				
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	



Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3					

…	B	X	X	X	Y	Y	Y	B	…

Example: TM for $\{0^n 1^n\}$

	0	**1**	**X**	**Y**	**B**
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	

...	B	X	X	X	Y	Y	Y	B	...

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
$* \ q_4$					

...	B	X	X	X	Y	Y	Y	B	...

Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	$-$	$-$	(q_3, Y, R)	$-$
q_1	$(q_1, 0, R)$	(q_2, Y, L)	$-$	(q_1, Y, R)	$-$
q_2	$(q_2, 0, L)$	$-$	(q_0, X, R)	(q_2, Y, L)	$-$
q_3	$-$	$-$	$-$	(q_3, Y, R)	(q_4, B, R)
$* \; q_4$	$-$	$-$	$-$	$-$	$-$

Example: TM for $\{0^n 1^n\}$

1.Find a 0, change it to X and move right

2.Move right to find an 1

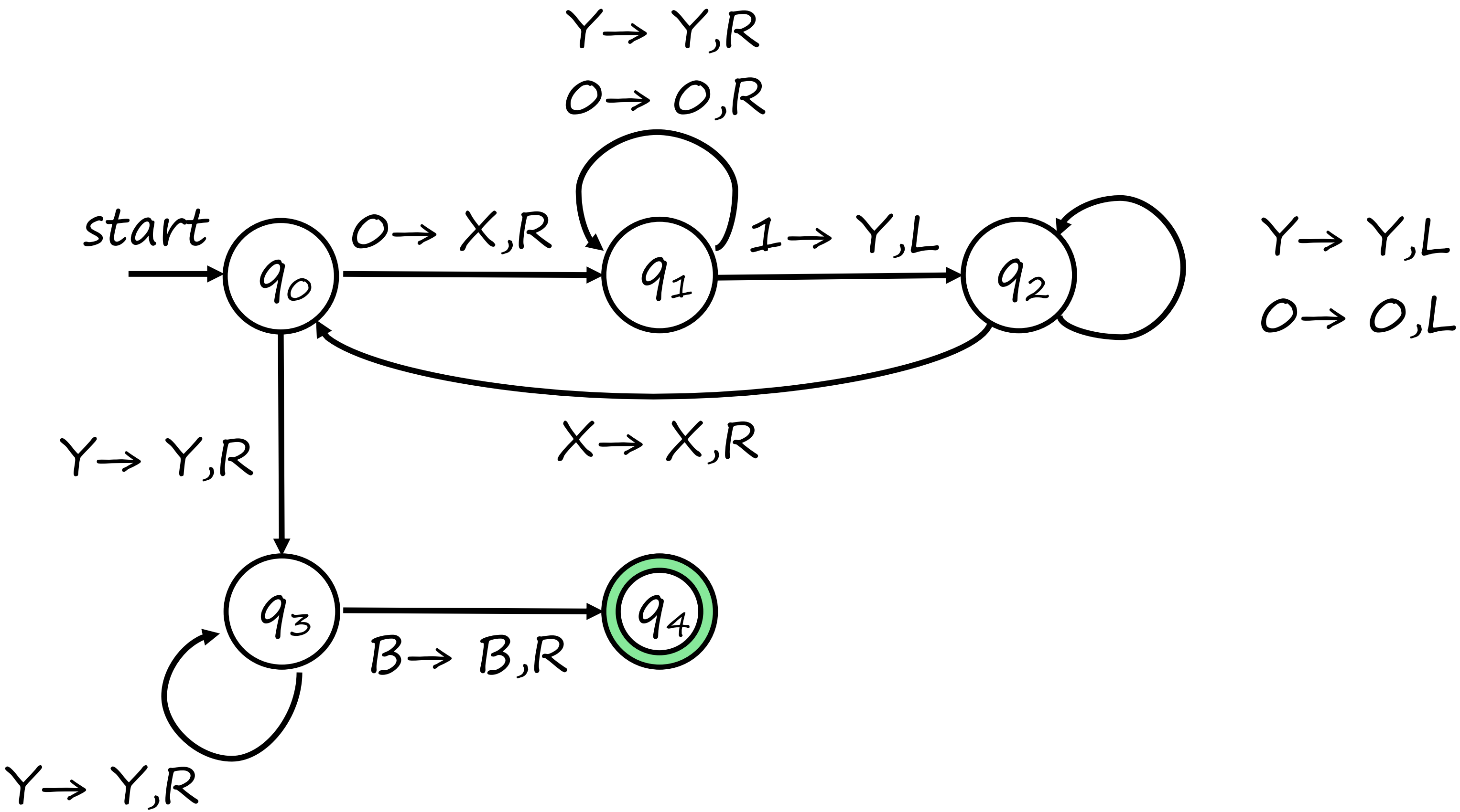
4.No 0 left, check if all symbol are Y

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	$-$	$-$	(q_3, Y, R)	$-$
q_1	$(q_1, 0, R)$	(q_2, Y, L)	$-$	(q_1, Y, R)	$-$
q_2	$(q_2, 0, L)$	$-$	(q_0, X, R)	(q_2, Y, L)	$-$
q_3	$-$	$-$	$-$	(q_3, Y, R)	(q_4, B, R)
$*\ q_4$	$-$	$-$	$-$	$-$	$-$

3.Move left to find an X

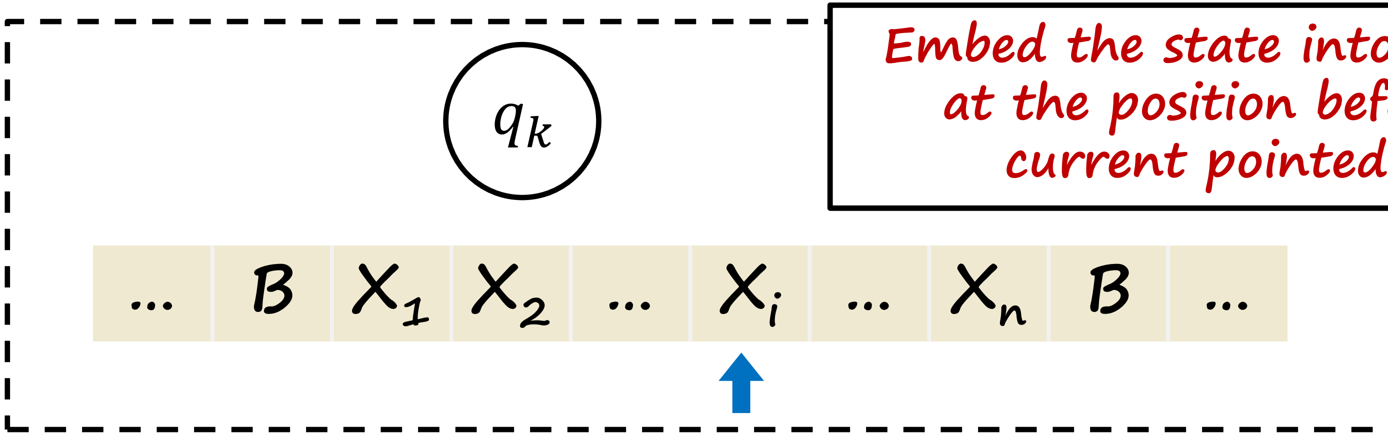
5.Accept when we meet a blank.

Example: TM for $\{0^n 1^n\}$



Instantaneous Descriptions for TM

- **Instantaneous Descriptions (ID's)** formally describe what a Turing machine does.
  - Tape contents, Current state, Location of type head



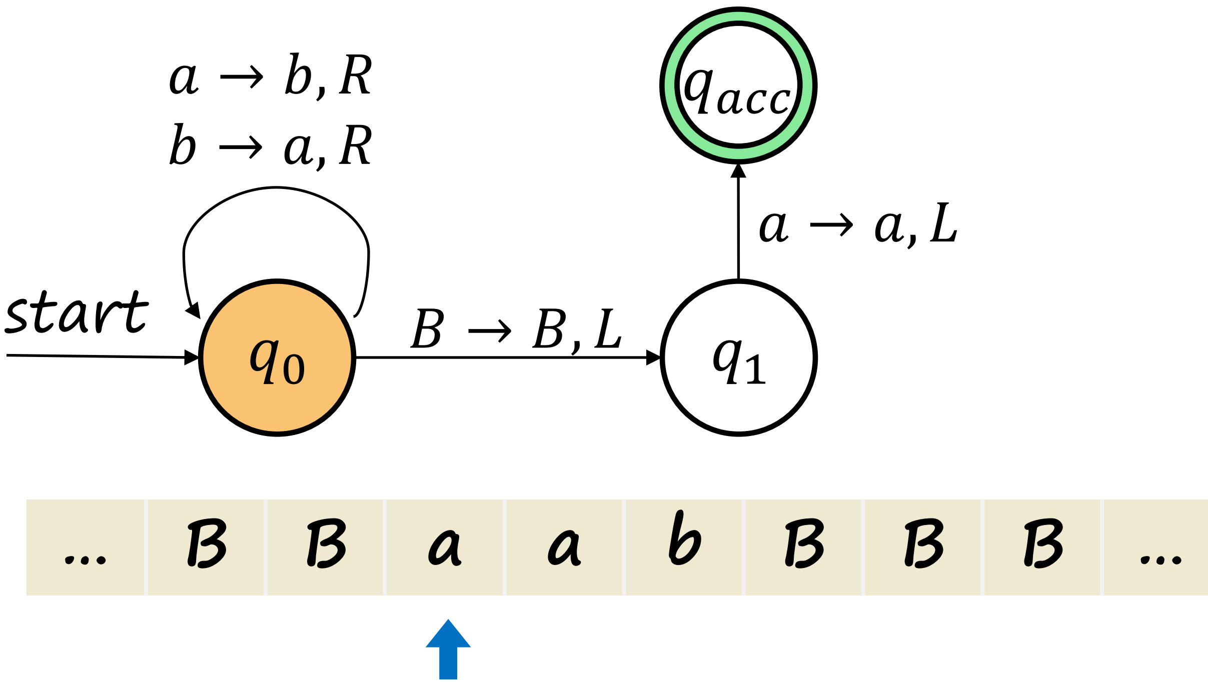
Embed the state into the tape at the position before the current pointed cell.

$$X_1X_2\ldots X_{i-1}q_kX_i\ldots X_n$$

Instantaneous Descriptions(ID's) for TM

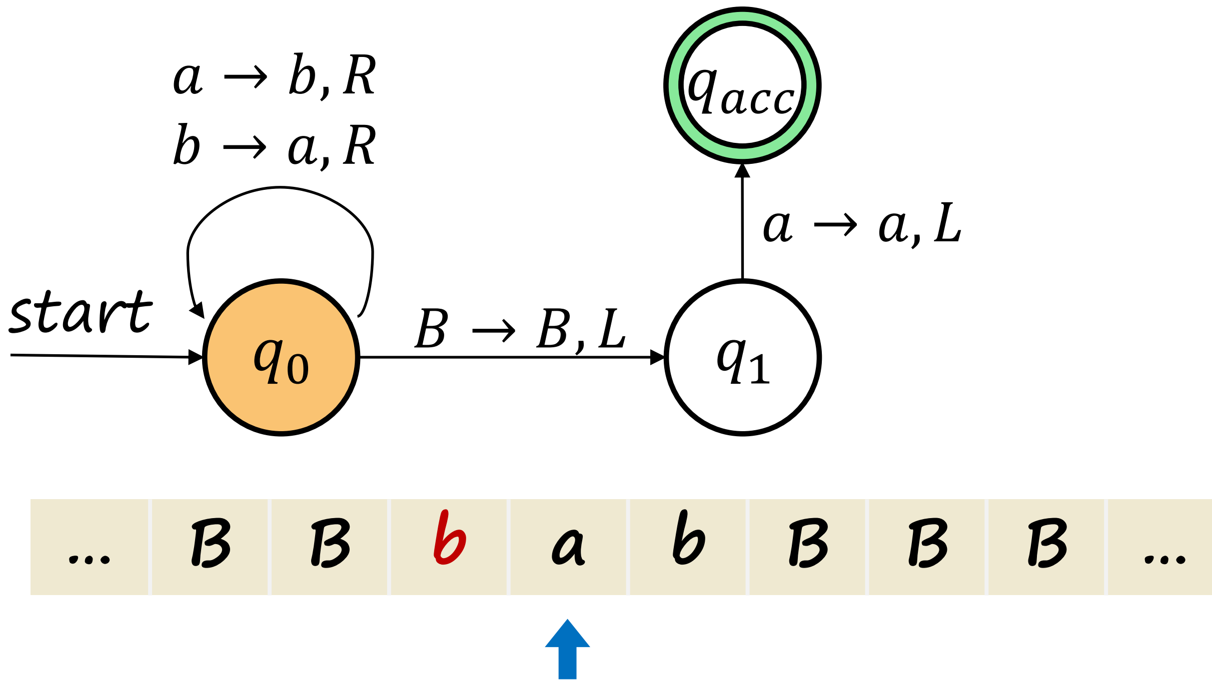
- Describe the moves of TM as $\vdash_M$
- If $\delta(q, X_i) = (p, Y, R)$ then
  - $X_1X_2 \dots X_{i-1}\boldsymbol{qX_i}X_{i+1} \dots X_n \ \vdash_M \ X_1X_2 \dots X_{i-1}\boldsymbol{Yp}X_{i+1} \dots X_n$
- If $\delta(q, X_i) = (p, Y, L)$ then
  - $X_1X_2 \dots X_{i-1}\boldsymbol{qX_i}X_{i+1} \dots X_n \ \vdash_M \ X_1X_2 \dots \boldsymbol{p}X_{i-1}\boldsymbol{Y}X_{i+1} \dots X_n$
- $\vdash_M^*$ indicated zero, one or more moves
  - $ID_1 \vdash_M^* ID_2$ means that ID_1 can be transformed to ID_2 through zero, one or more moves.

ID's and Moves



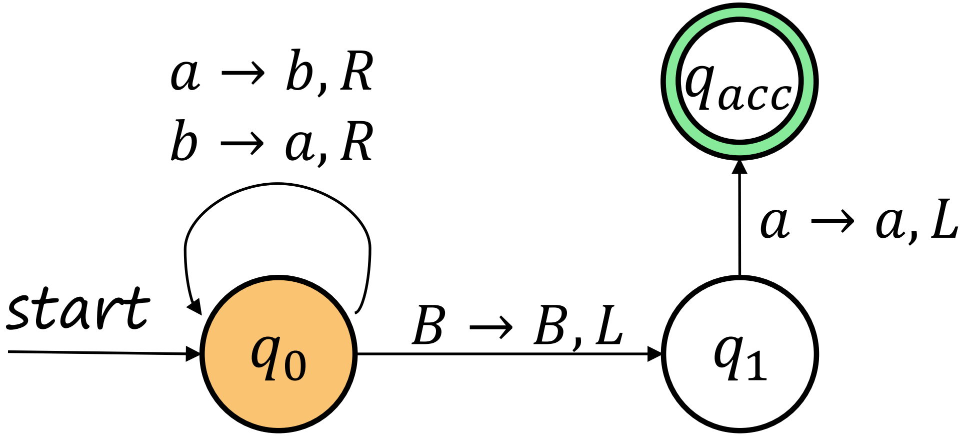
q_0aab

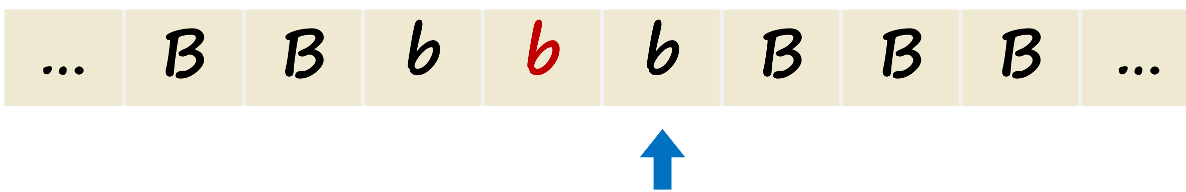
ID's and Moves



$$q_0aab \vdash_M bq_0ab$$

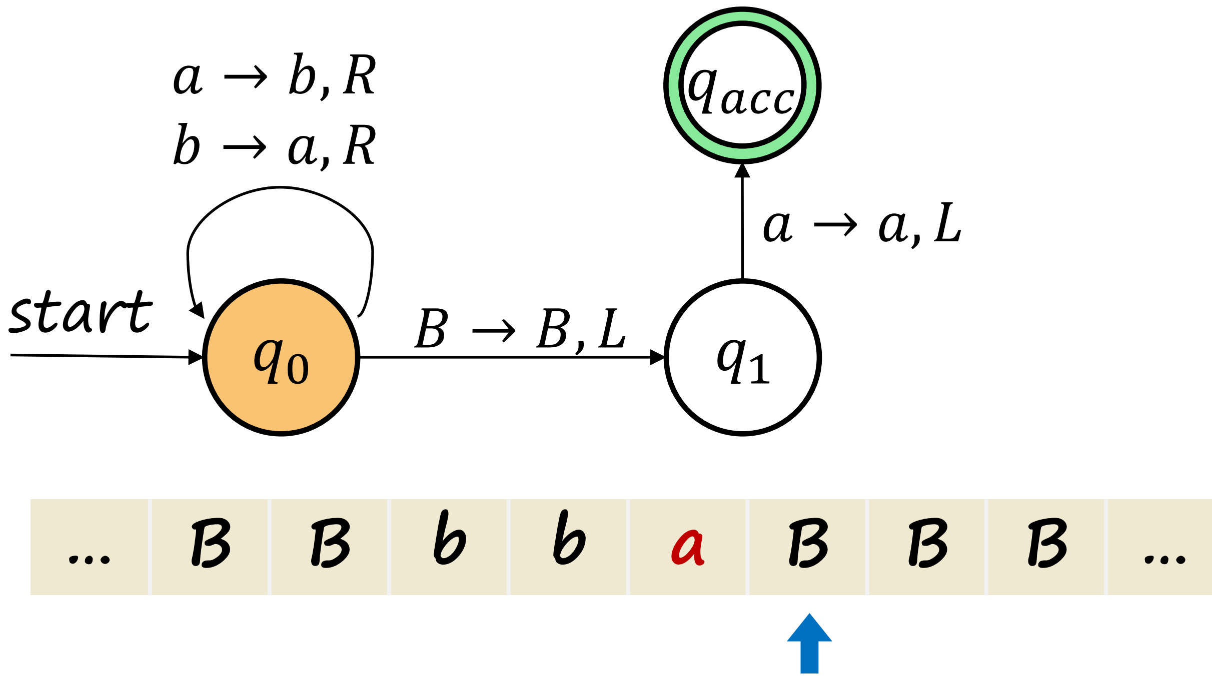
ID's and Moves





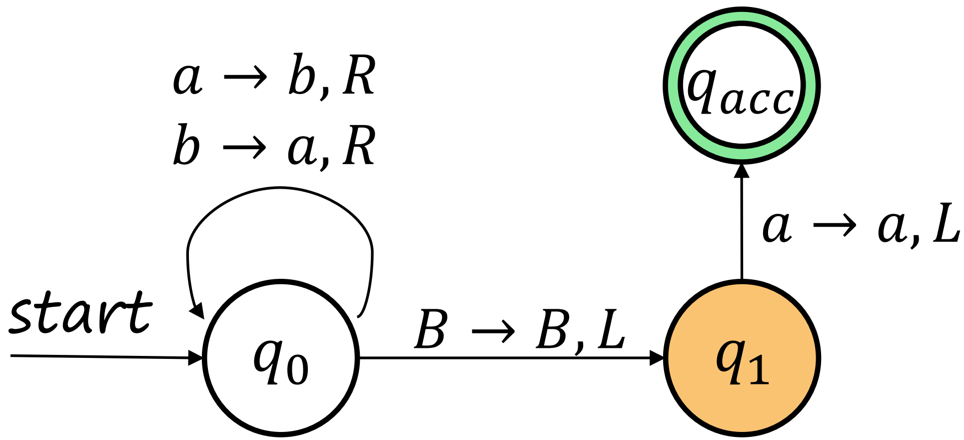
$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b$

ID's and Moves



$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M$
$bbaq_0B$

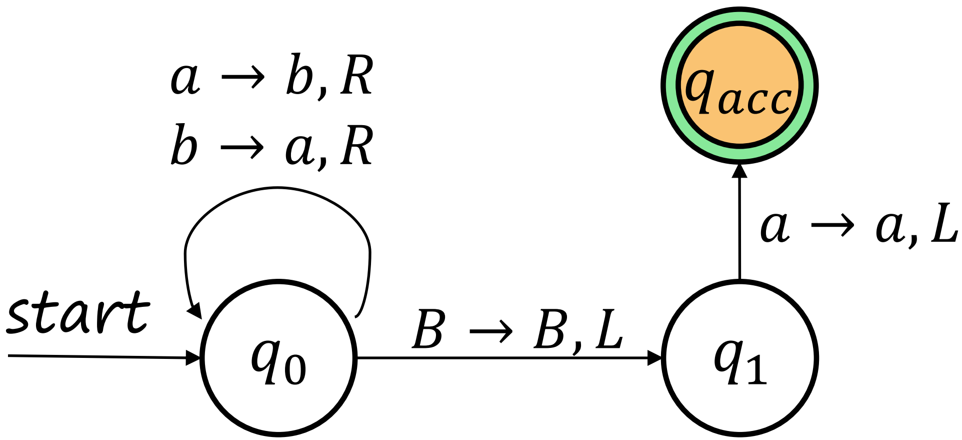
ID's and Moves

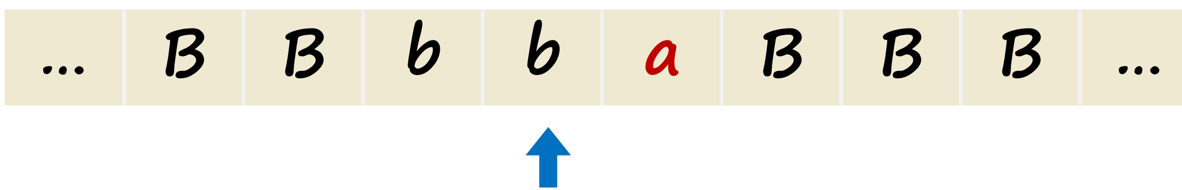


...	B	B	b	b	a	B	B	...

$$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M bbaq_0B \vdash_M bbq_1aB$$

ID's and Moves





$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b \vdash_M$
$bbaq_0B \vdash_M bbq_1aB \vdash_M bq_{acc}ba$

The Language of a TM

- Similar to DFA, the language of a TM $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{\Sigma}, \boldsymbol{\Gamma}, \boldsymbol{\delta}, \boldsymbol{q_0}, \boldsymbol{B}, \boldsymbol{F})$ is set of strings that it accepts, or:

$$L(\boldsymbol{M}) = \{w | w \in \Sigma^* \land p \in F \land q_0 w \vdash^* \alpha p \beta\}$$

- For a certain language L, if there exists a TM $\boldsymbol{M}$ such that $L = L(\boldsymbol{M})$, then we call L is a **recursively enumerable language(递归可枚举语言)**， or ***RE***.

Computation Capability

- For any certain type of automata (e.g. DFA,TM), the computation capability of it can be regarded as all the languages the type of automata can accept.

$$P(T) = \{L(A) | A \ is \ a \ T\},$$

where T can be DFA , Turing Machine, etc.

- For DFA, the capability is all the regular languages. For Turing Machine, it is all the *RE* languages.
  - $P(DFA) = \{L | L \ is \ a \ Regular \ Language\}$
  - $P(TM) = \{L | L \ is \ a \ RE \ Language\}$

Computation Capability

- For two types of automata T_1 and T_2
  - If $P(T_1) = P(T_2)$, we say T_1 and T_2 have same capability.
  - If $P(T_1) \subset P(T_2)$, we say T_2 have greater capability than T_1.
- DFA and TM:
  - For any language $\boldsymbol{L}$ accepted by a DFA ($\boldsymbol{L} \in \boldsymbol{P}(\boldsymbol{DFA})$), there's a TM $\boldsymbol{M}$ such that $L(\boldsymbol{M}) = \boldsymbol{L}$ ($\boldsymbol{L} \in \boldsymbol{P}(\boldsymbol{TM})$), so it means that $\boldsymbol{P}(\boldsymbol{DFA}) \subseteq \boldsymbol{P}(\boldsymbol{TM})$
  - And we know there's some languages can be accepted by TM but can not be accepted by DFA(like $\{1^n2^n\}$), so we have $P(DFA) \subset P(TM)$, which means **TM have greater computation capability than DFA**

Church-Turing Thesis

Every effective method of computation is either equivalent to or weaker than a Turing machine.

- This is not a **theorem** but a falsifiable scientific hypo**thesis**. But it has been thoroughly tested! So we have strong faith in its correctness.
- This means Turing Machine can model any “computation”.

~~What problems can a computer solve?~~

↓

~~What languages can a computer recognize?~~

↓

What languages can TM recognize?

How we investigate computability?

2.1 Discussion on Problems

Part1. Intro & Set theory(I) : Basics & Formal Language.

Part2. Set Theory(II): Axiom system & Cardinality.

Part3. Capture Structures: Binary Relation & Function

2.2 Discussion on Computation

Part1. Turing Machine Basics.

Part2. Variants of Turing Machine. Church-Turing Thesis.

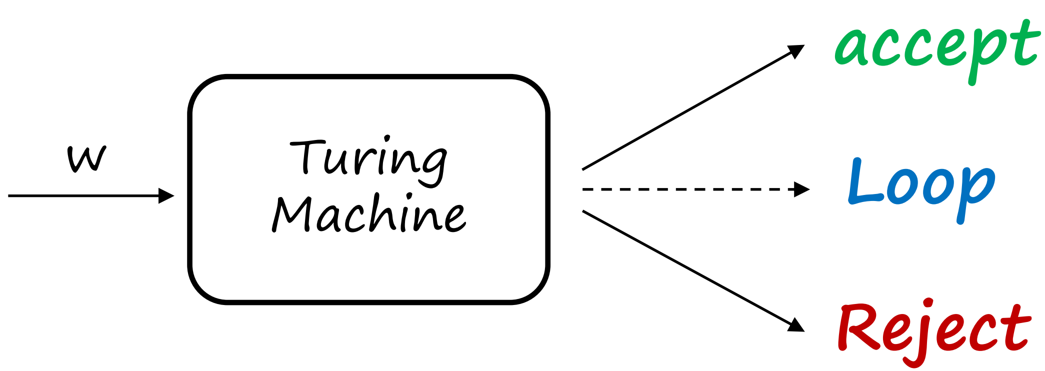
2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

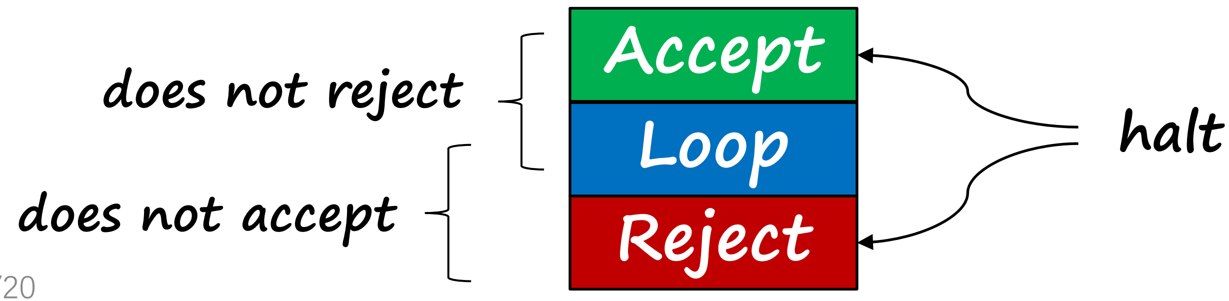
Output of TM

- For a certain input string, the output of a TM can be one in three kinds:
  - The TM **accepts** this string.
  - The TM **rejects** this string.
  - Rather than accepting or rejecting, the TM **loops** forever on the input string and never stops.



Very Important Terminology

- Let M be a Turing machine and let s be a string.
  - M **accepts** s if it enters an accepting state when running on s
  - M **rejects** s if it enters a rejecting state when running on s.
  - M **loops infinitely** on s when running on s if it enters neither an accepting nor a rejecting state.
  - M **does not accept** s if it either rejects s or loops on s.
  - M **does not reject** s if it either accepts s or loops on s.
  - M **halts** on s if it accepts w or rejects s.



More Details of RE

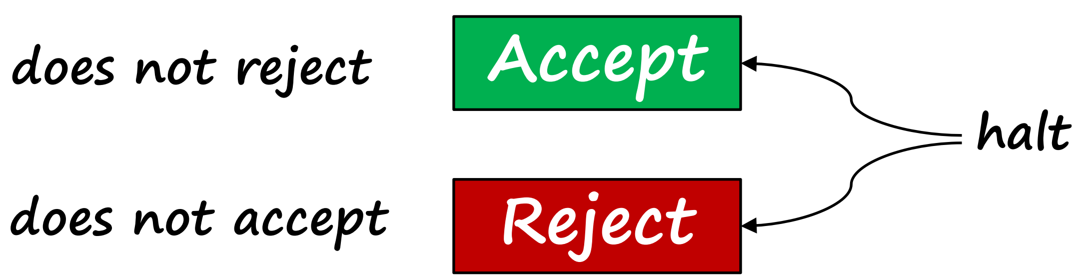
- We've known that for a certain language L, if there exists a TM $\boldsymbol{M}$ such that $L = L(\boldsymbol{M})$, then we call L is a **recursively enumerable language(递归可枚举语言)**，or ***RE***.
- So for any RE $\boldsymbol{L}$, a TM $\boldsymbol{M}$ that $L(M) = L$, and any string $\boldsymbol{w}$
  – If $w \in L, M$ accepts w in finite steps.
  – If $w \notin L, M$ does not accept w, it means M rejects w or loops forever
- We also call the TM $\boldsymbol{M}$ a **recognizer** for the language $\boldsymbol{L}$, and $\boldsymbol{L}$ is **Turing-recognizable(图灵可识别的).**

More Details of RE

- In other words, as an recognizer, $\boldsymbol{M}$ will tell you correct on every correct input, but $\boldsymbol{M}$ is not necessary to tell you wrong on every wrong input.
  - You can not determine the input is correct or wrong until $\boldsymbol{M}$ halts.
- But a problem is solvable(computable) if the computation process finishes in finite steps.

Decider

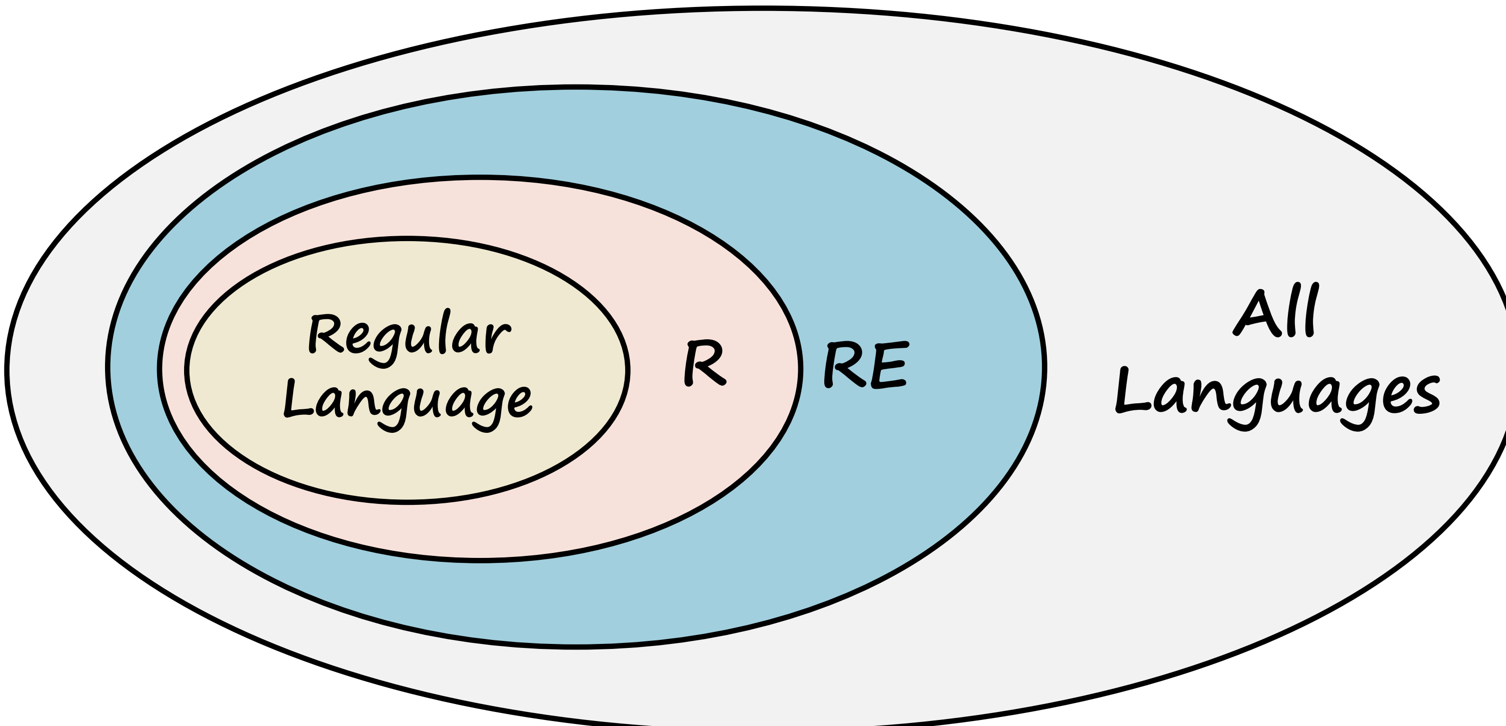
- Some Turing machines always halt; they never go into an infinite loop.
- If $\boldsymbol{M}$ is a TM and $\boldsymbol{M}$ halts on every possible input, then we say that $\boldsymbol{M}$ is a decider.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



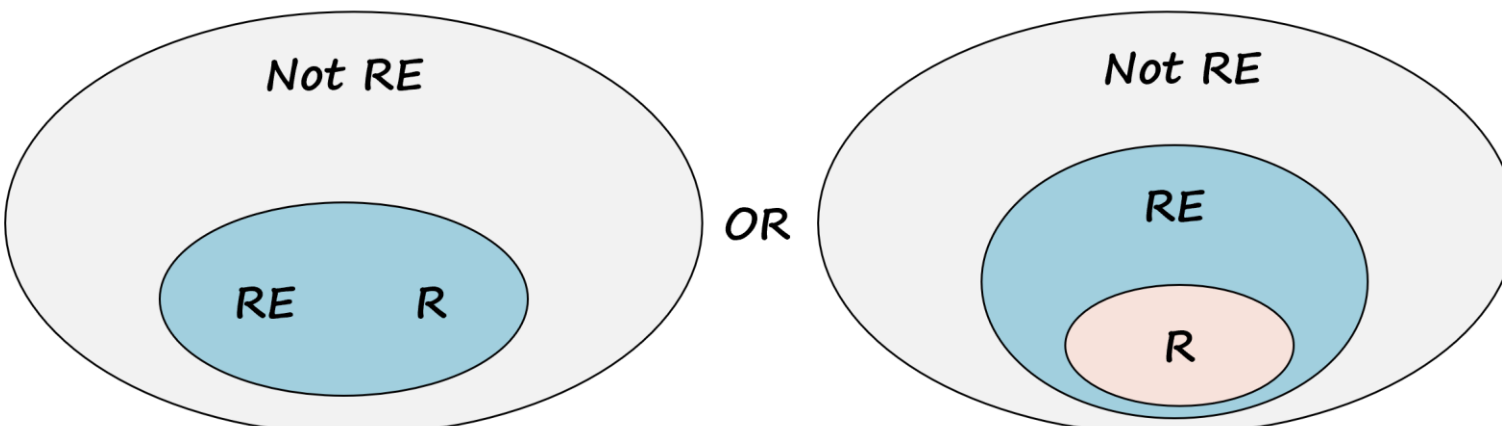
Recursive Language

- A language $\boldsymbol{L}$ is **recursive(递归的)** if there's a TM $\boldsymbol{M}$ such that:
  - If $w \in L, M$ accepts w in finite steps.
  - If $w \notin L, M$ rejects w in finite steps.
- We also call the TM $\boldsymbol{M}$ a **decider** for the language $\boldsymbol{L}$, and $\boldsymbol{L}$ is **Turing-decidable(图灵可判定的).**
- Decidable problems, in some sense, are that can definitely be "solved" by a computer.
- Recursive Language is a subset of RE language

Language Hierarchy



Question:
RE=R?



Next

How to model the computation process.

- The equivalence of TM and an ideal computer.
- Church-Turing thesis.