Turing Machine II

Zeyu Mi

2023-12-20

Adapted From:
IALC 8.2,8.3.1,9.1,9.2

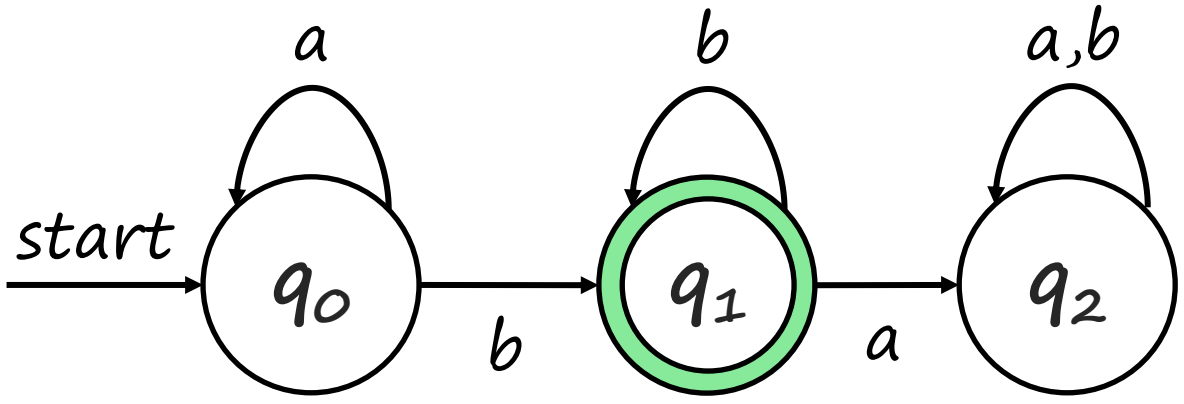
Review

- **Function**
  - Inverse of function.
- **Computers are complex.**
- **Automata is a mathematical model of a computing device**
  - Clean and simple
  - DFA: Five tuples $(Q, \Sigma, \delta, q_0, F)$

Review：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
→ 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***

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

$\hat{\delta}$ 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.

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

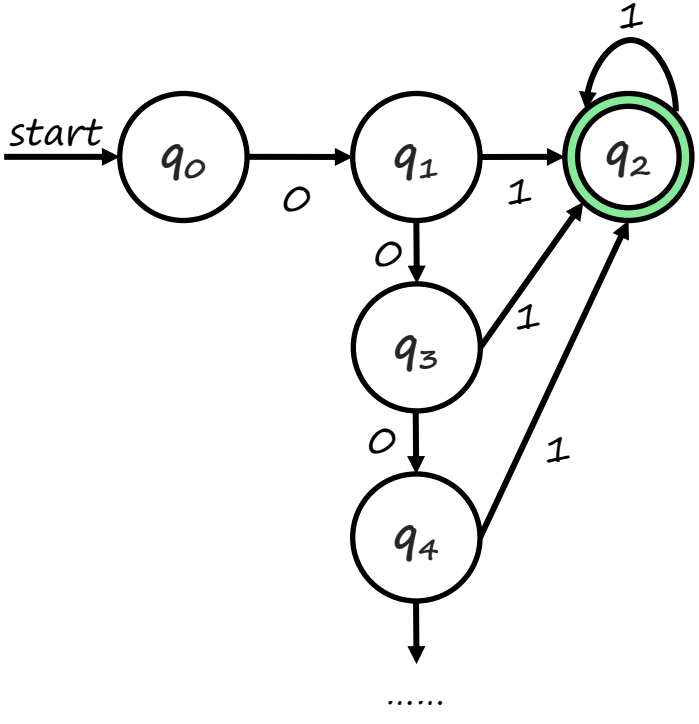
- For a certain language L, if there exists a DFA $\boldsymbol{A}$ such that $L = L(\boldsymbol{A})$, then we call L is a **regular language(正则语言)**.

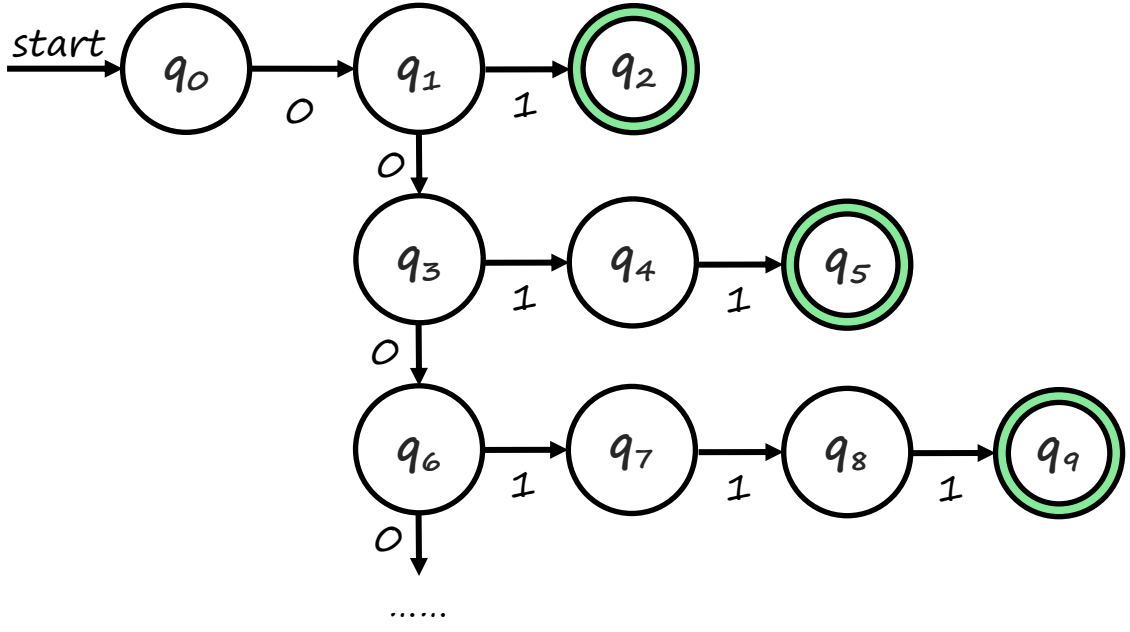
We'll not take much time in DFA and regular language. Feel free to read chapter 2,3,4 in IALC if you are interested in it.

But DFA can not recognize all the languages.

The Limitation of DFA

- There is no DFA for languages like $\{0^n 1^n | n \in \mathbb{N}^+\}$.





The Limitation of DFA

- There is no DFA for languages like $\{0^n 1^n | n \in \mathbb{N}^+\}$.
- We need a more powerful “machine” to model our computer.

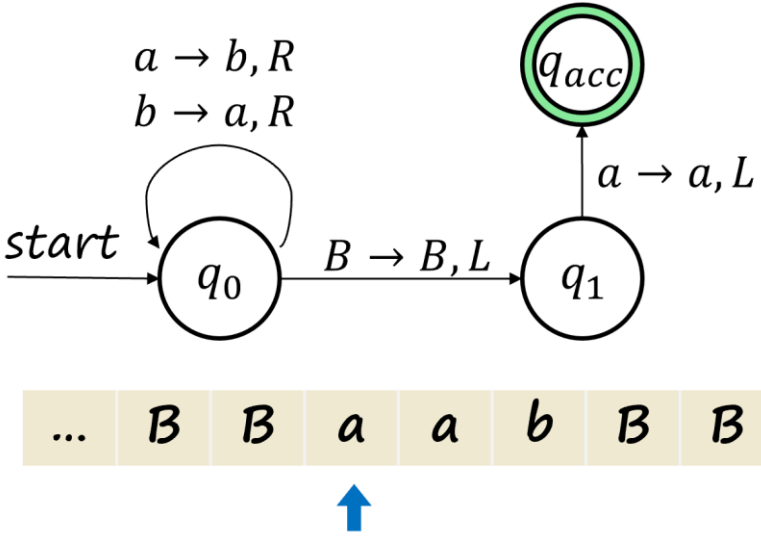
Turning Machine!!

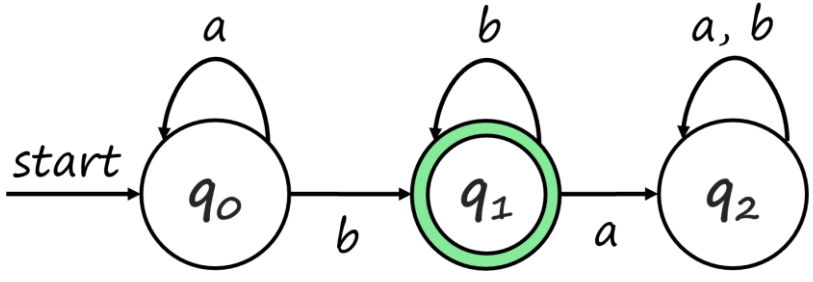
Turing Machine

- 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





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

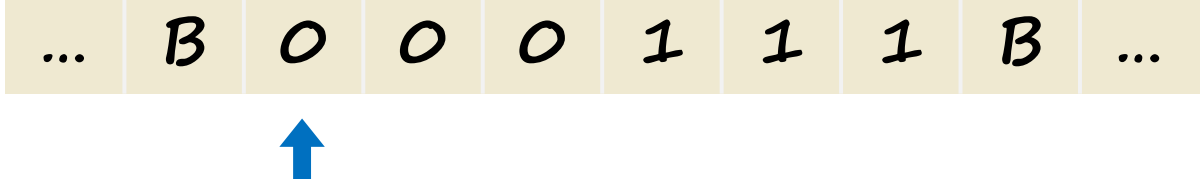
- Count the number of 0 and the number of 1, and compare them.
  - But we have no idea how to store the number and compare them.
- Or when we see a 0, we try to find an 1, and repeat the process until we count every symbol.

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

- 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.

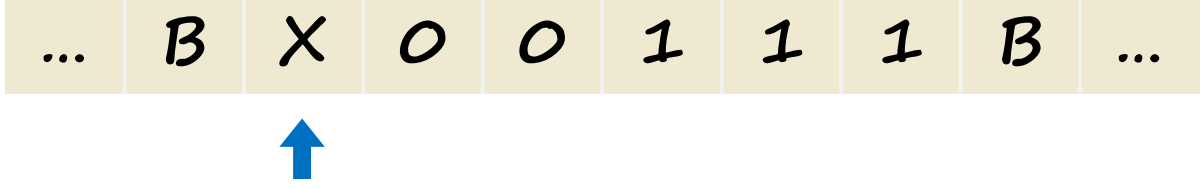
Example: TM for $\{0^n 1^n\}$

We've found a 0



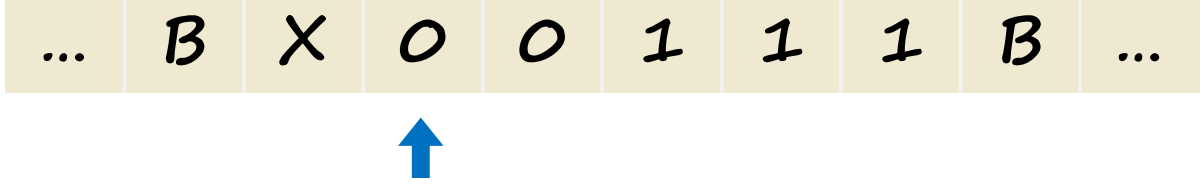
Example: TM for $\{0^n 1^n\}$

Change it to X



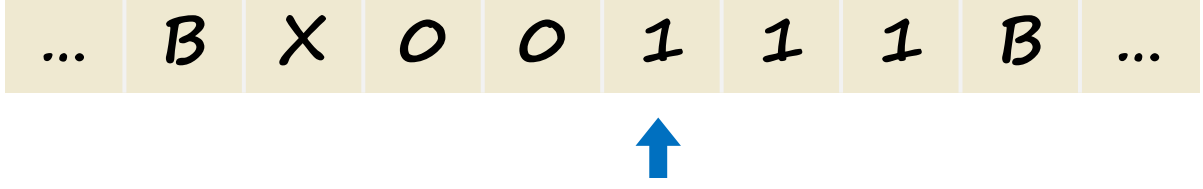
Example: TM for $\{0^n 1^n\}$

Move right to find an 1



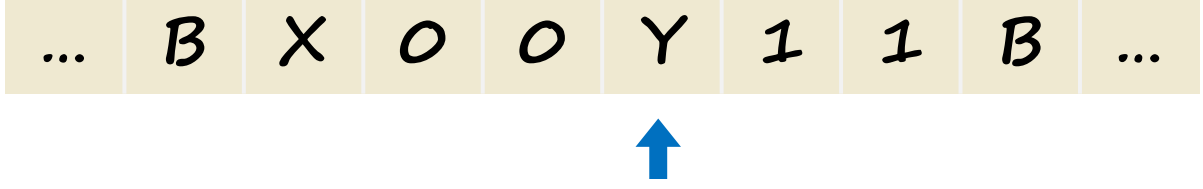
Example: TM for $\{0^n 1^n\}$

We've found an 1



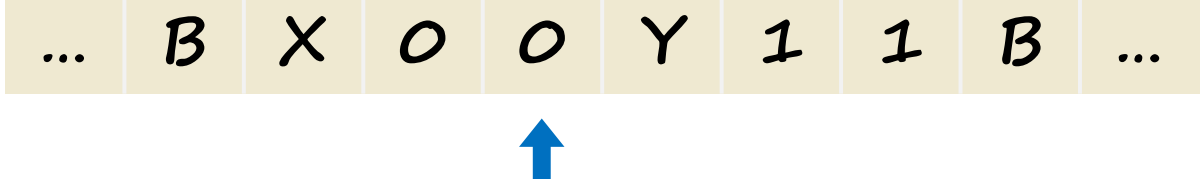
Example: TM for $\{0^n 1^n\}$

Change it to Y



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

Turn back to find 0 at the beginning



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

Stop move left when we meet X.

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

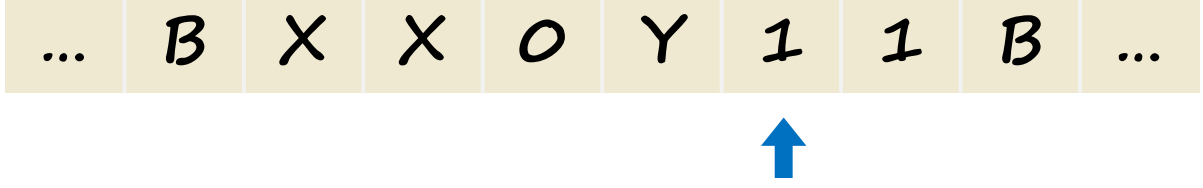
Take a right move. We found another 0.

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

Change it to X and continue.

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

Another 1.



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

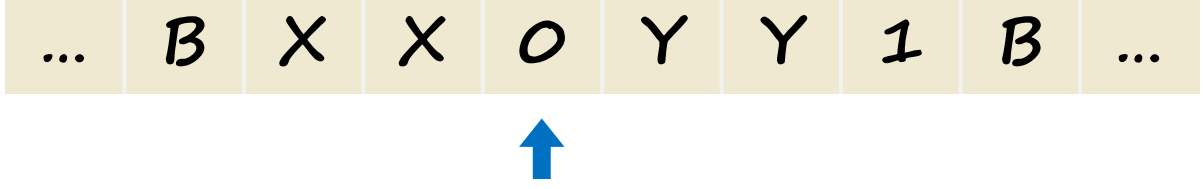
Change it to Y and turn back again.

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

Meet X, stop moving left.

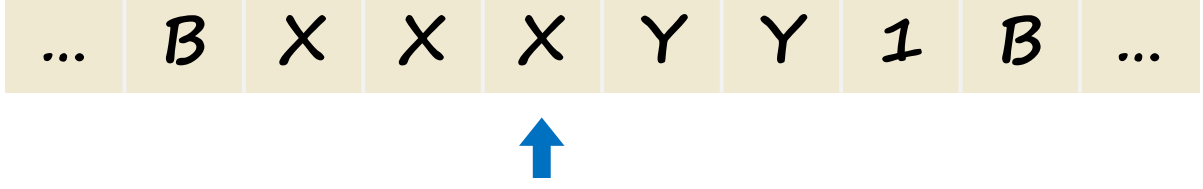
Example: TM for $\{0^n 1^n\}$

Another 0.



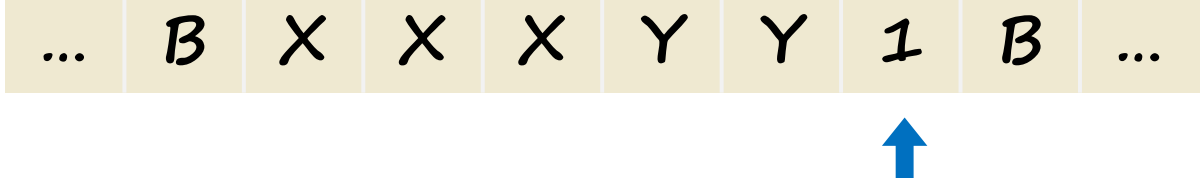
Example: TM for $\{0^n 1^n\}$

Change it to X.



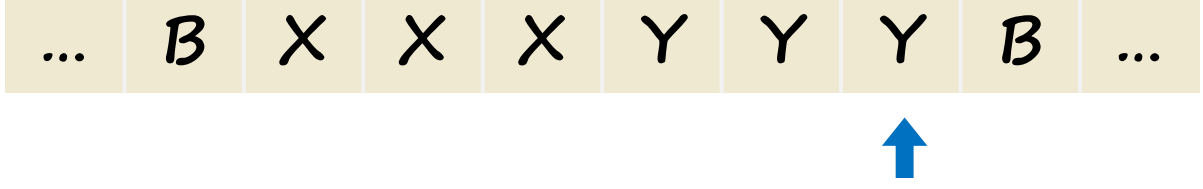
Example: TM for $\{0^n 1^n\}$

Another 1.



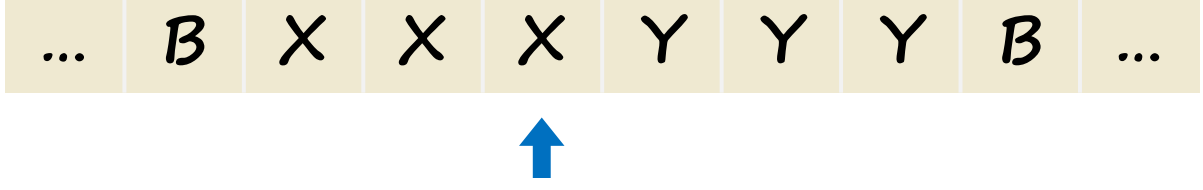
Example: TM for $\{0^n 1^n\}$

Change it and turn back again.



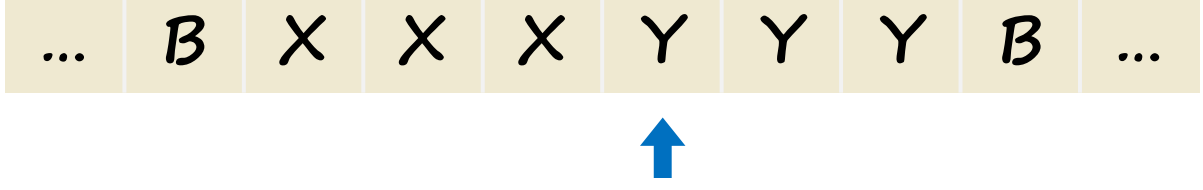
Example: TM for $\{0^n 1^n\}$

Meet X, take a right move.



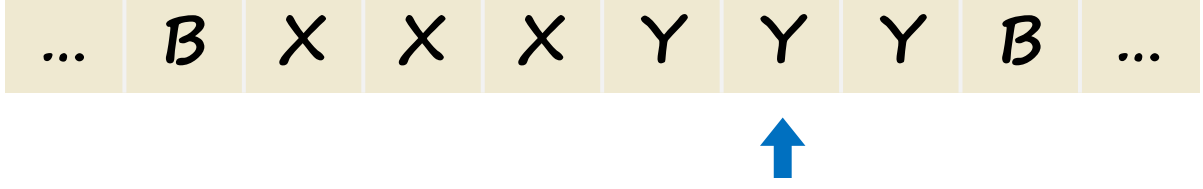
Example: TM for $\{0^n 1^n\}$

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



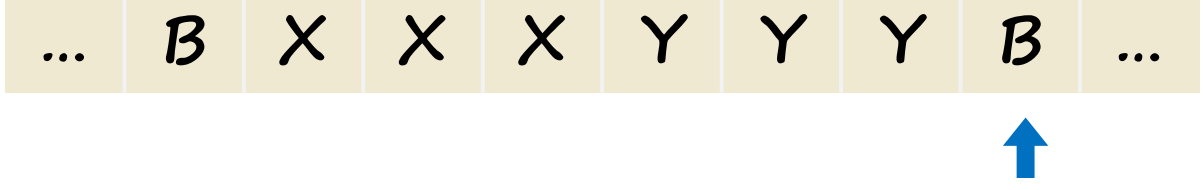
Example: TM for $\{0^n 1^n\}$

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



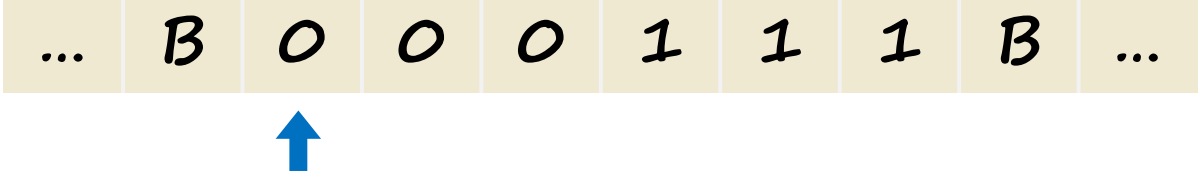
Example: TM for $\{0^n 1^n\}$

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



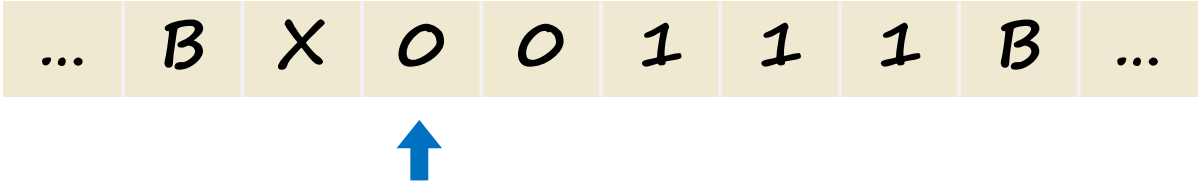
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ q_0					



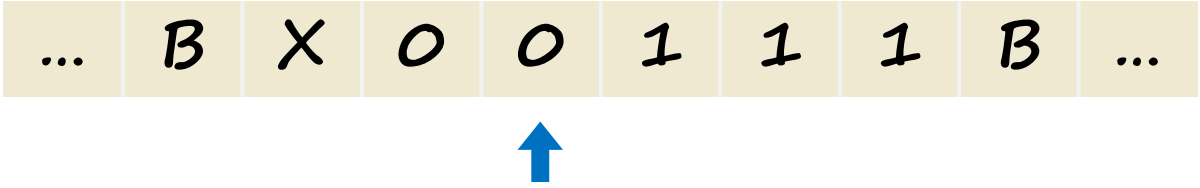
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ q_0	(q_1, X, R)				
q_1					



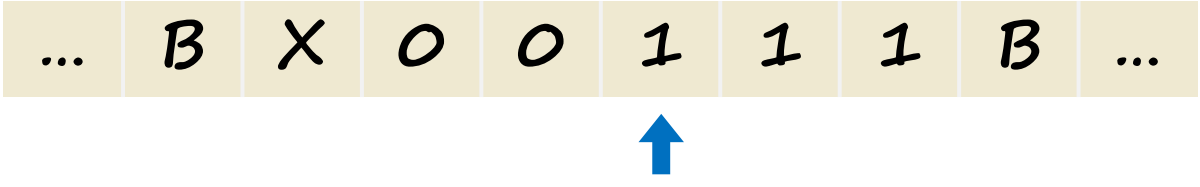
Example: TM for $\{0^n 1^n\}$

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



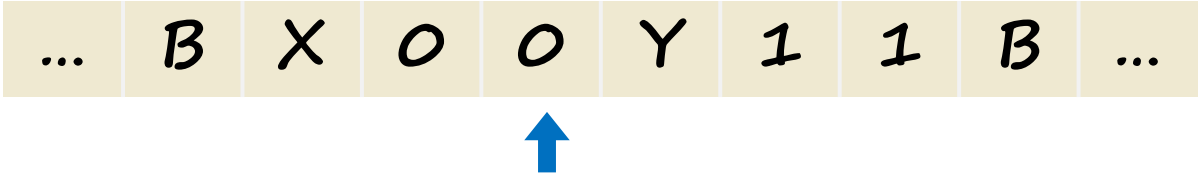
Example: TM for $\{0^n 1^n\}$

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



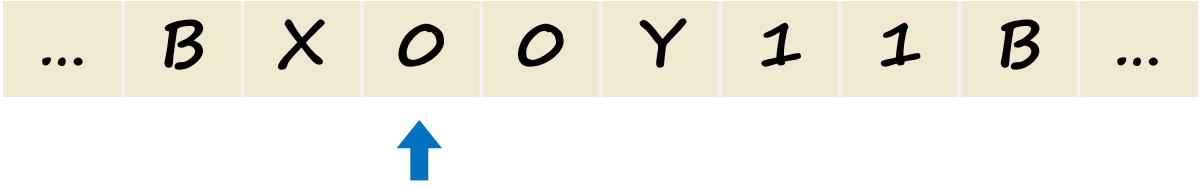
Example: TM for $\{0^n 1^n\}$

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



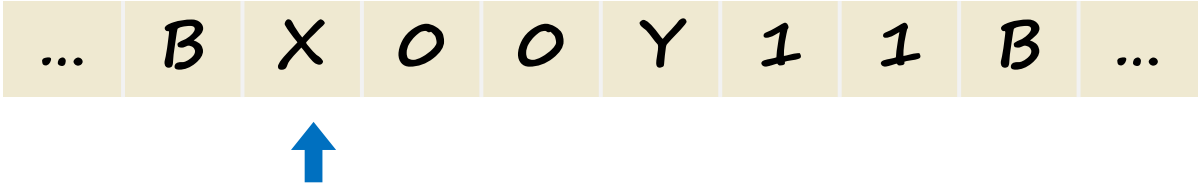
Example: TM for $\{0^n 1^n\}$

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



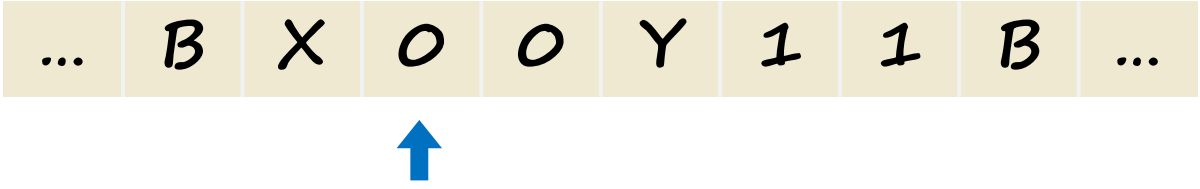
Example: TM for $\{0^n 1^n\}$

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



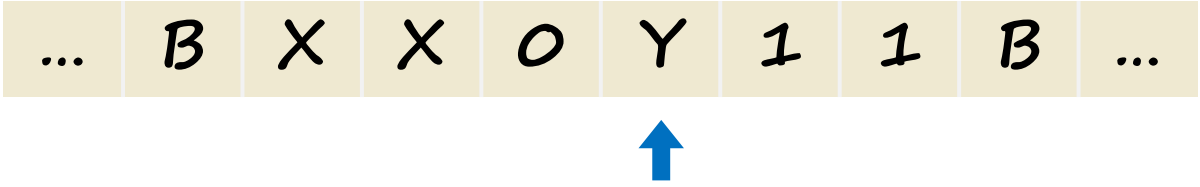
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)		



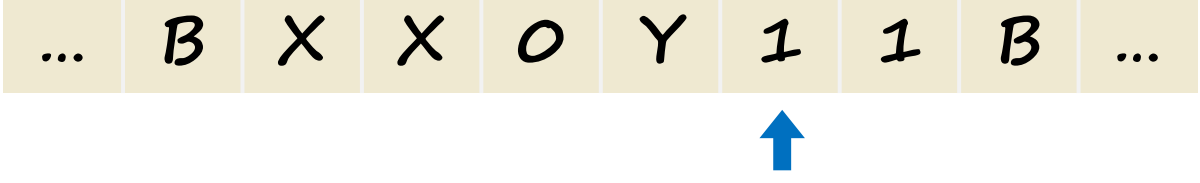
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)		



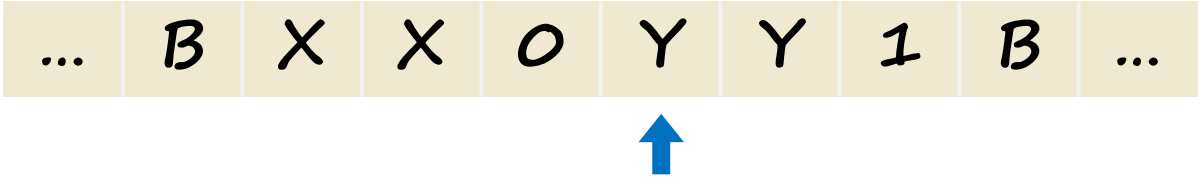
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)		



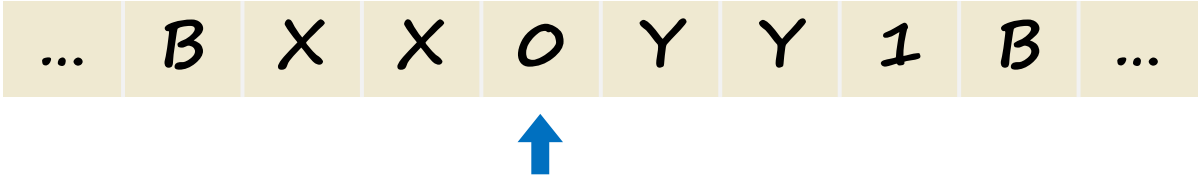
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)		



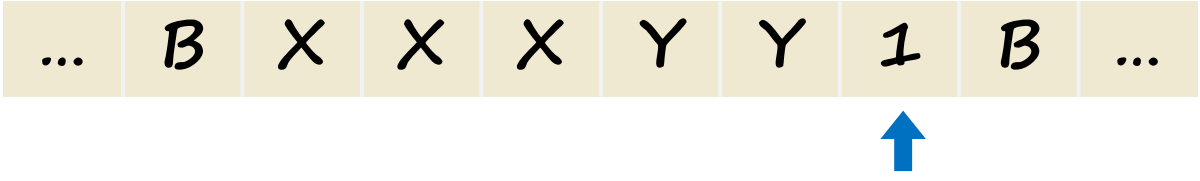
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)	



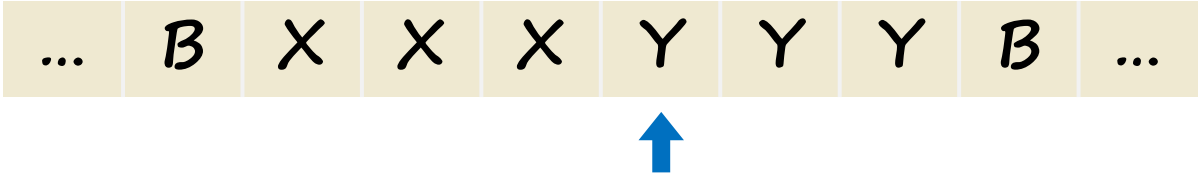
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)	



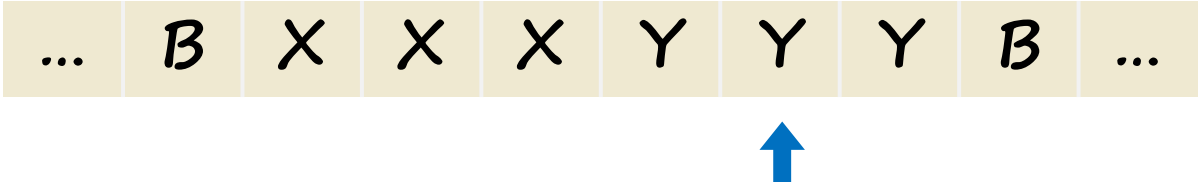
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)	



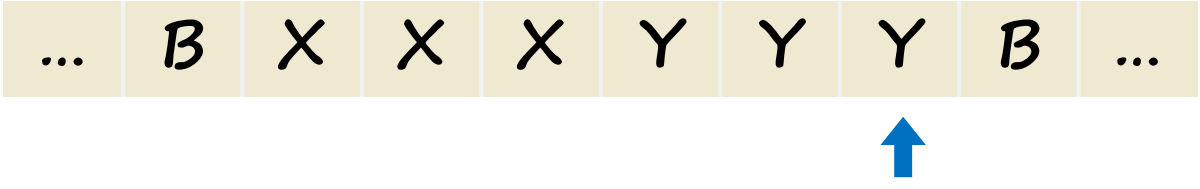
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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					



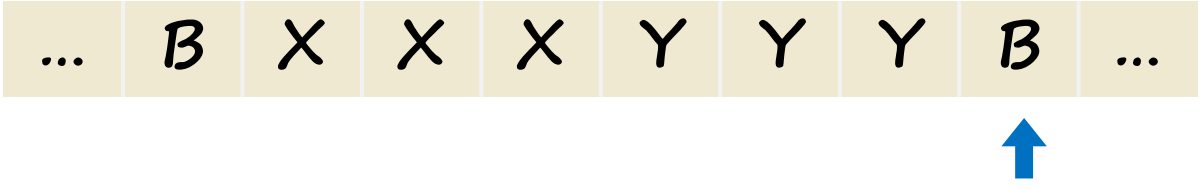
Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
→ 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)	



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

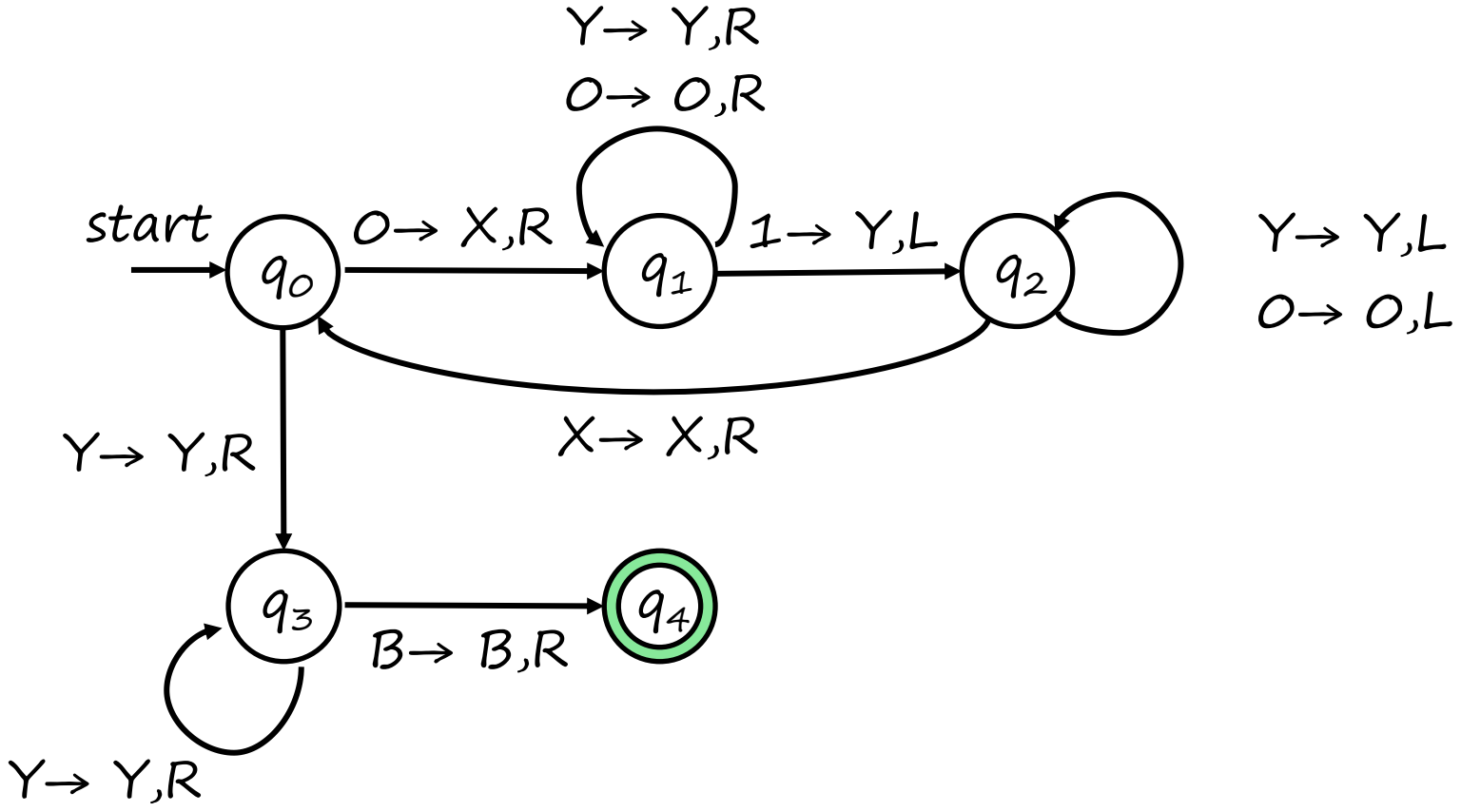
	0	1	X	Y	B
→ q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
* q_4					



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

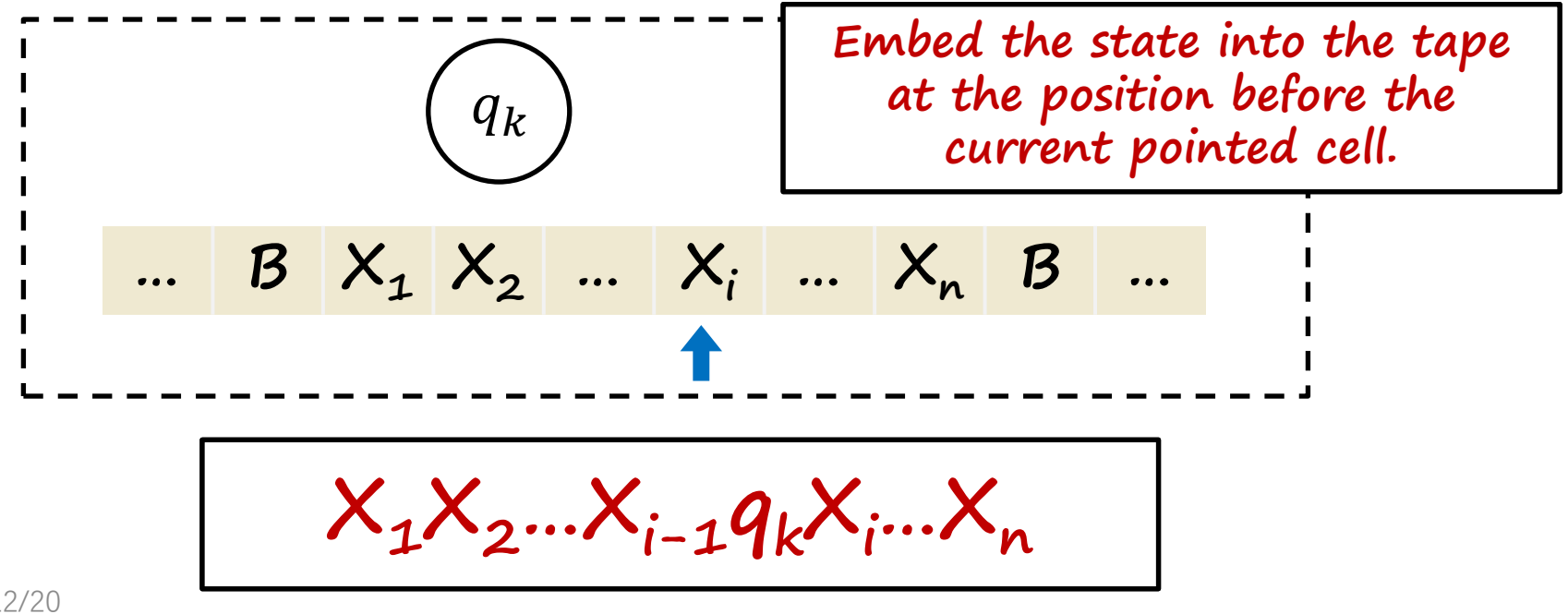
	0	1	X	Y	B
→ q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
* q_4	–	–	–	–	–

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



Instantaneous Descriptions for TM

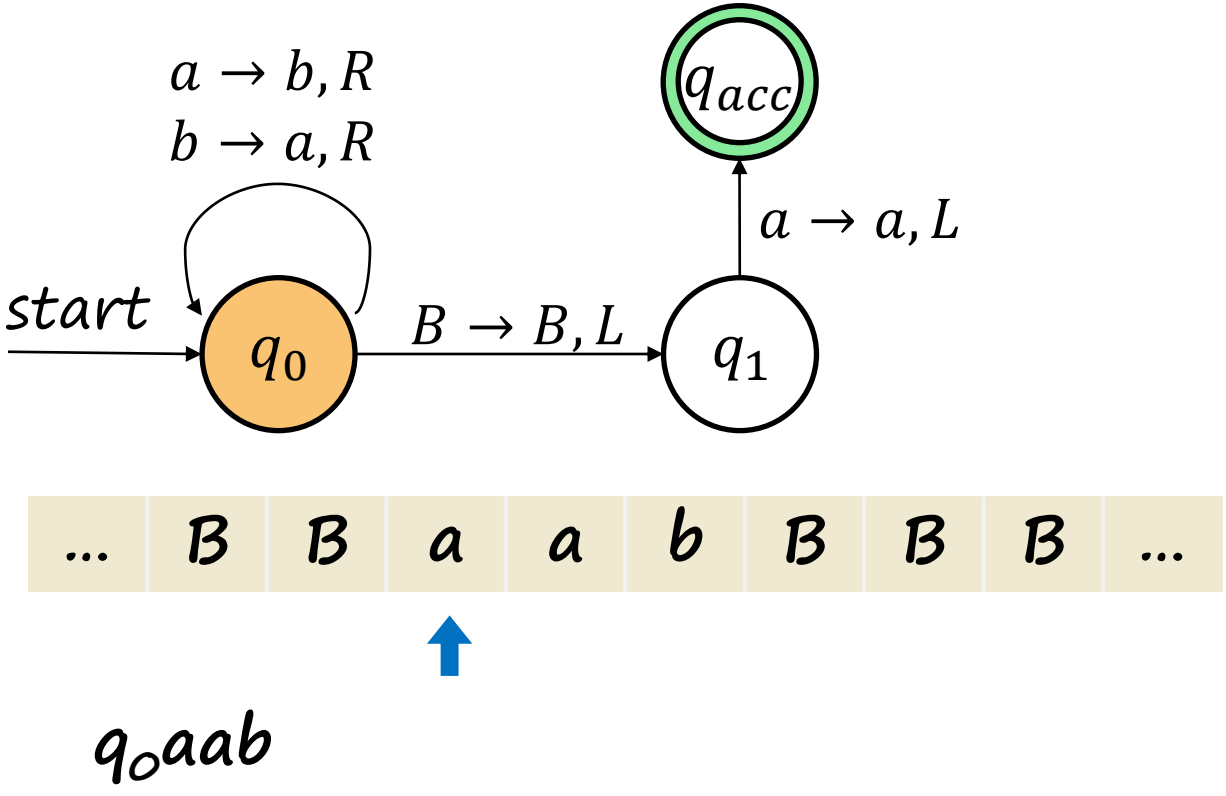
- **Instantaneous Descriptions (ID's)** formally describe what a Turing machine does.
  - Tape contents, Current state, Location of type head



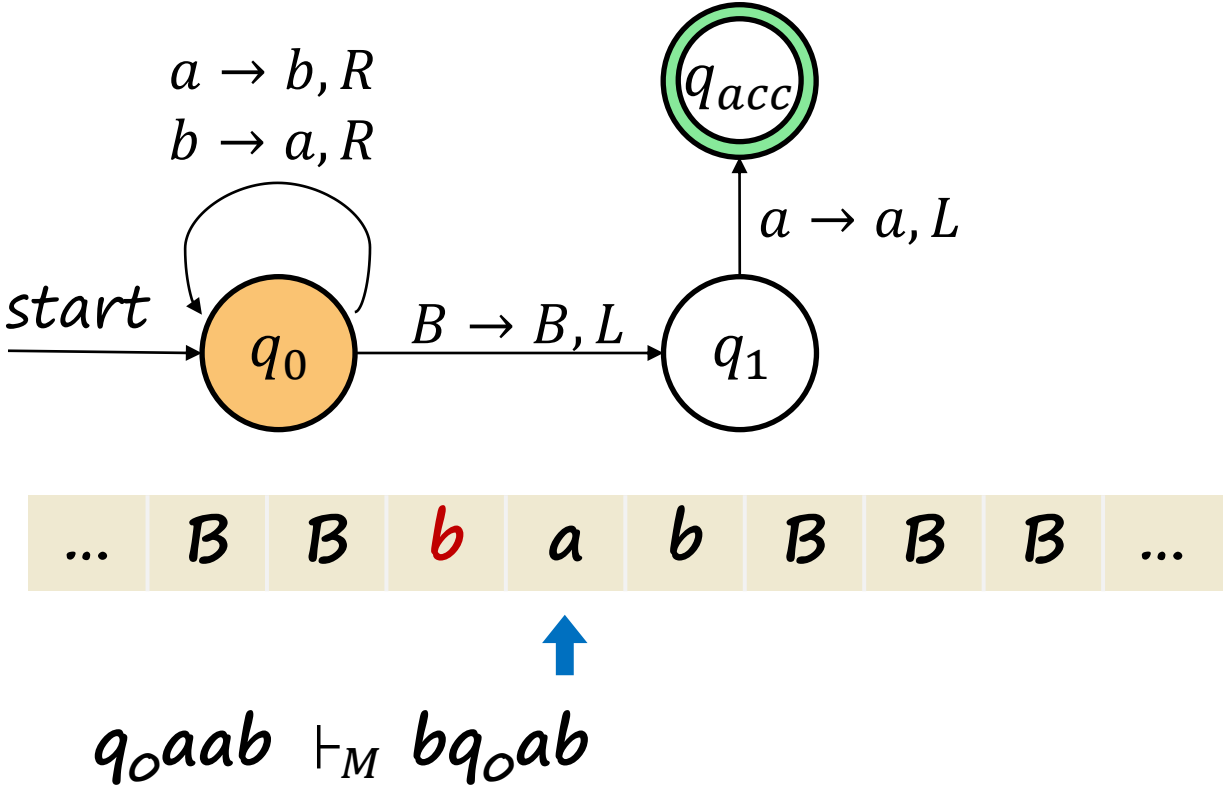
Instantaneous Descriptions(ID's) for TM

- Describe the moves of TM as $\vdash_M$
- If $\delta(q, X_i) = (p, Y, R)$ then
  - $X_1 X_2 \ldots X_{i-1} \boldsymbol{q X_i} X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-1} \boldsymbol{Y p} X_{i+1} \ldots X_n$
- If $\delta(q, X_i) = (p, Y, L)$ then
  - $X_1 X_2 \ldots X_{i-1} \boldsymbol{q X_i} X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots \boldsymbol{p} X_{i-1} \boldsymbol{Y} X_{i+1} \ldots X_n$
- $\vdash_M^*$ 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.

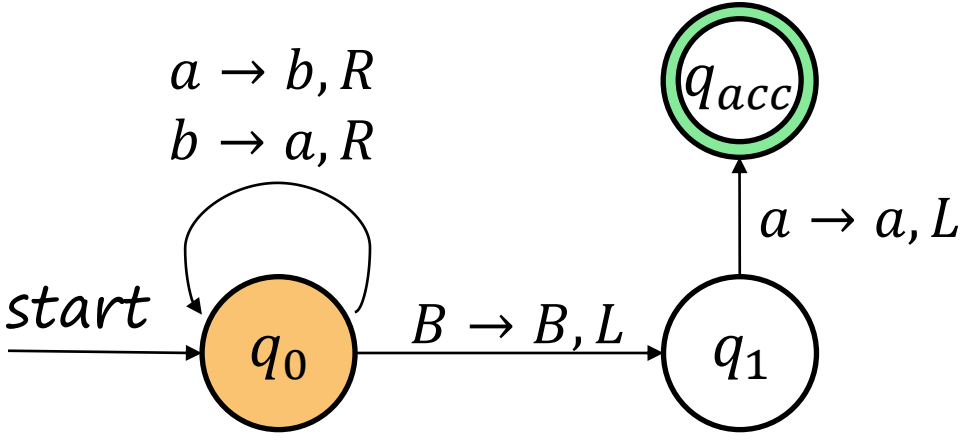
ID's and Moves



ID's and Moves



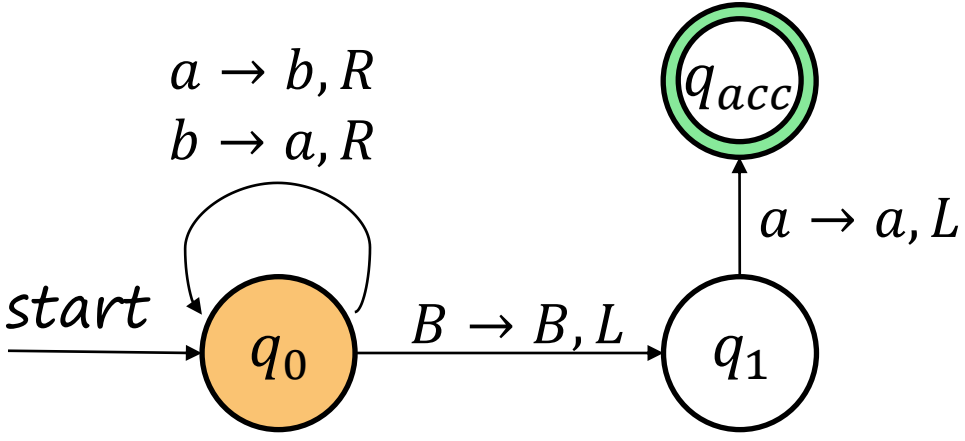
ID's and Moves



...	B	B	b	b	b	B	B	B	...

$q_0aab \vdash_M bq_0ab \vdash_M bbq_0b$

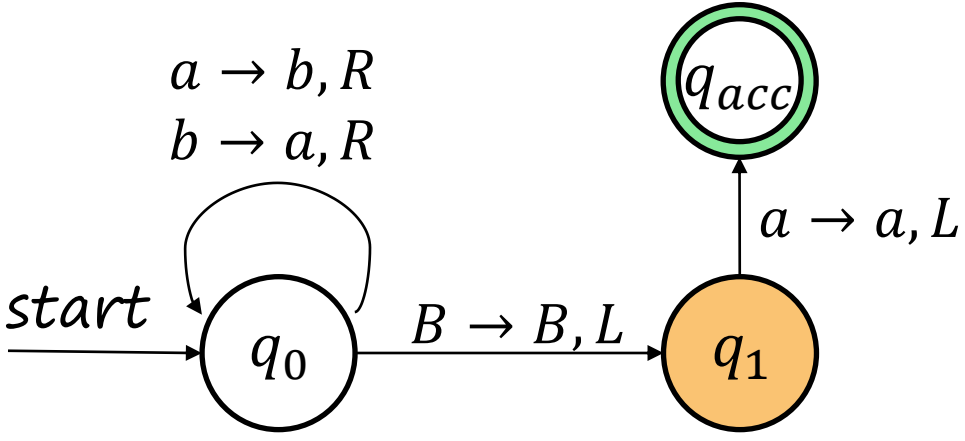
ID's and Moves



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

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

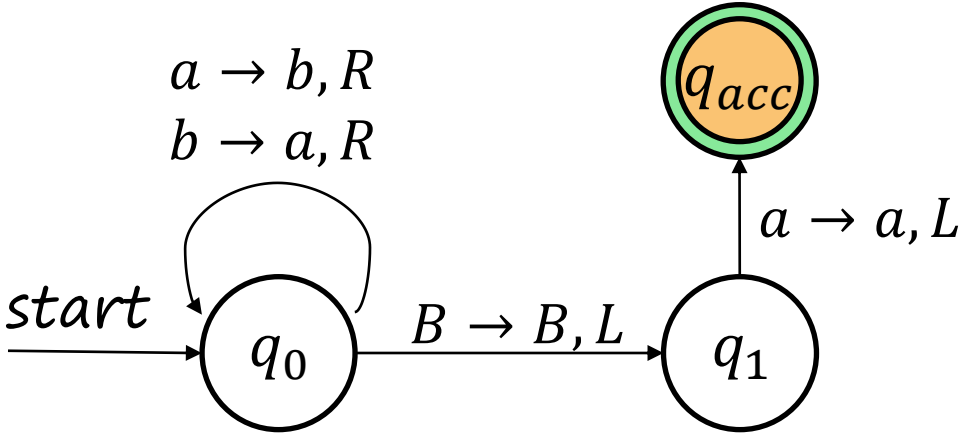
ID's and Moves



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

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

ID's and Moves



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

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

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:

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

- 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***.

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 ($\boldsymbol{L} \in \boldsymbol{P(DFA)}$), there's a TM $\boldsymbol{M}$ such that $L(\boldsymbol{M}) = \boldsymbol{L}$ ($\boldsymbol{L} \in \boldsymbol{P(TM)}$), so it means that $\boldsymbol{P(DFA) \subseteq P(TM)}$
  - 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.

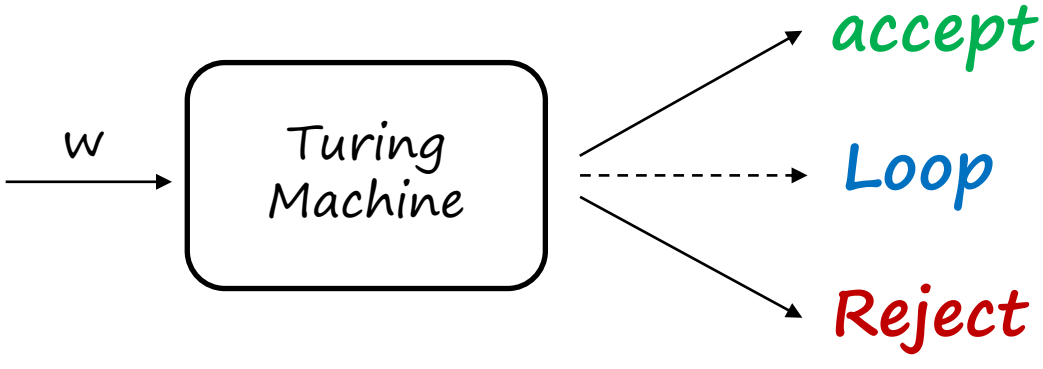
2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

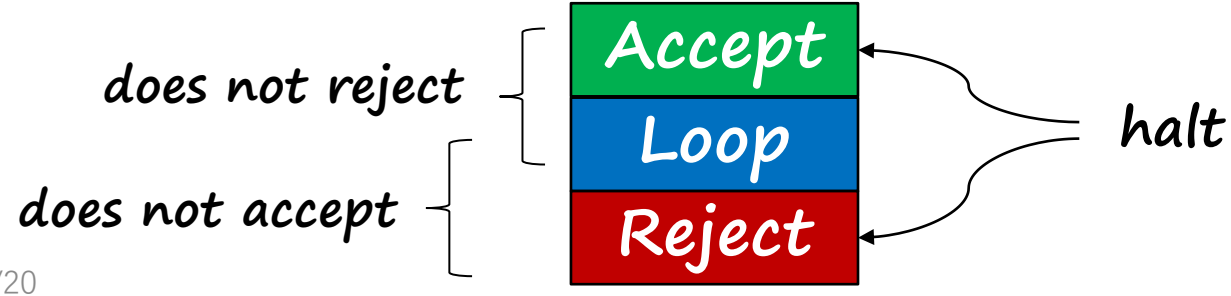
Output of TM

- 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

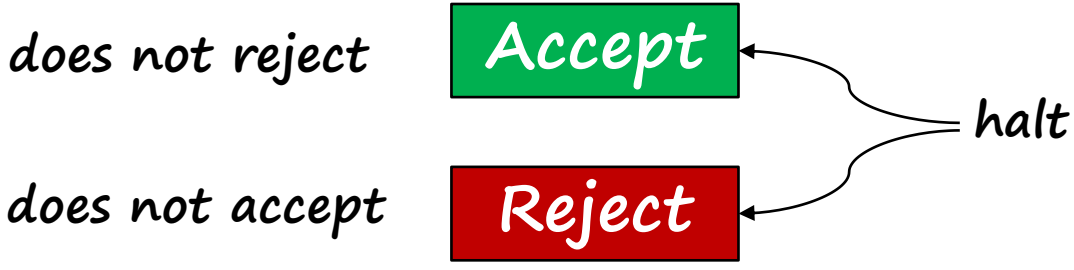
- We've known that for a certain language L, if there exists a TM $\boldsymbol{M}$ such that $L = L(\boldsymbol{M})$, then we call L is a **recursively enumerable language(递归可枚举语言)**, or ***RE***.
- So for any RE $\boldsymbol{L}$, a TM $\boldsymbol{M}$ that $L(M) = L$, and any string $\boldsymbol{w}$
  - 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

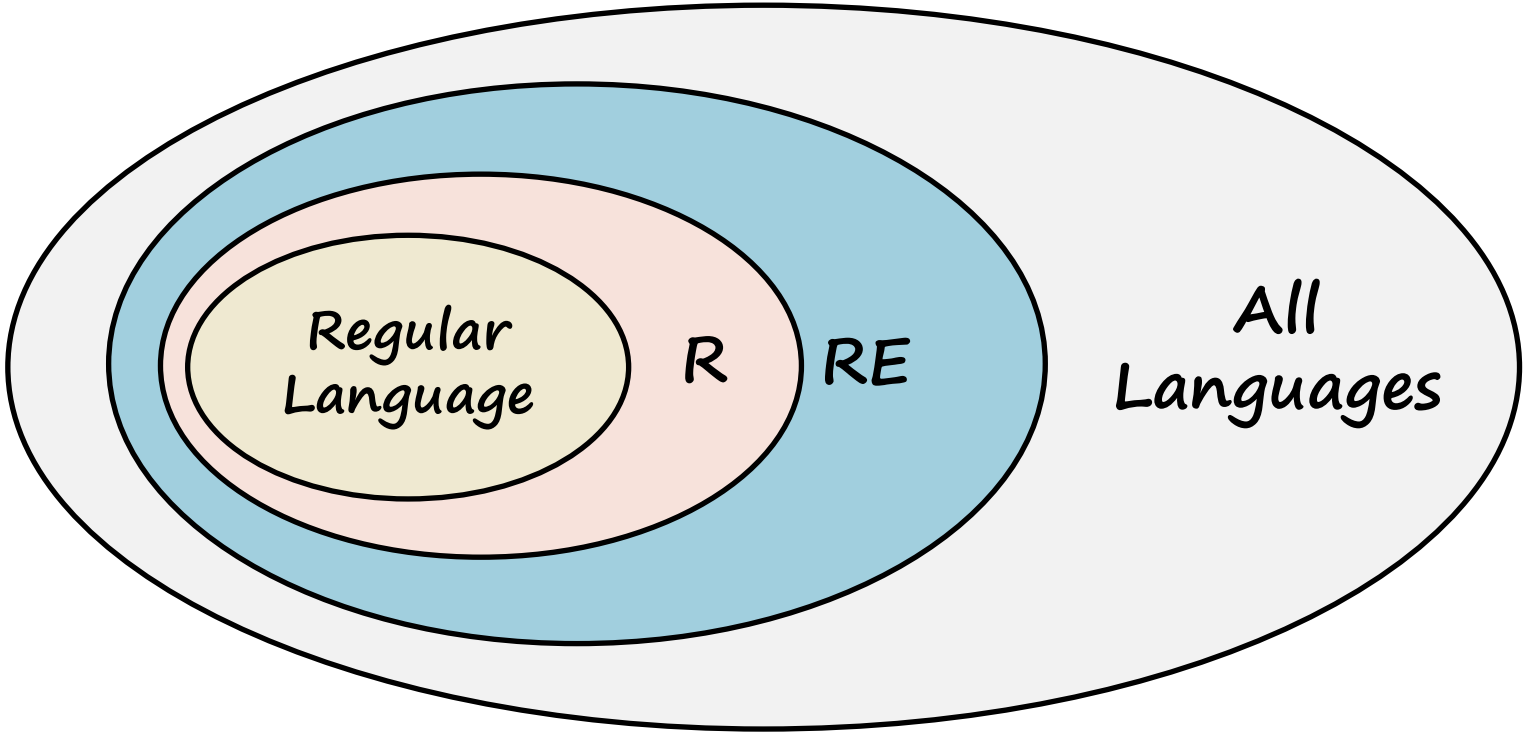
- Some Turing machines always halt; they never go into an infinite loop.
- If M is a TM and M halts on every possible input, then we say that M is a decider.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



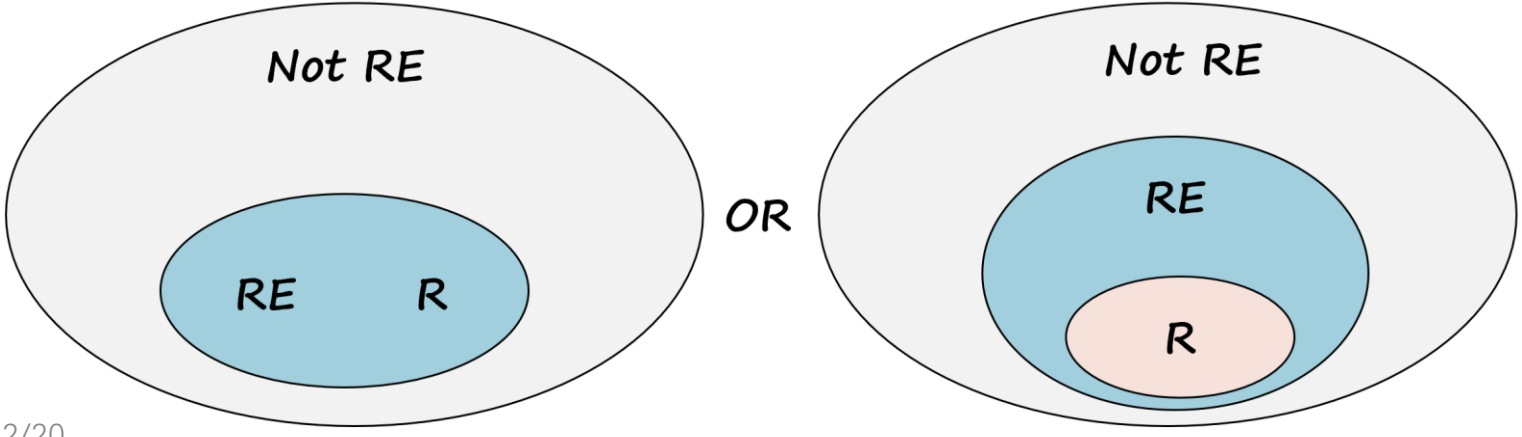
Recursive Language

- 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?



Universal Turing Machine

通用图灵机

An Observation

- When we've been discussing Turing machines, we've talked about designing specific TMs to solve specific problems.
  - TM for 0^n1^n
- Does this match your real-world experiences? Do you have one computing device for each task you need to perform?
- Or can we make a "**reprogrammable** Turing machine?"

A TM Simulator

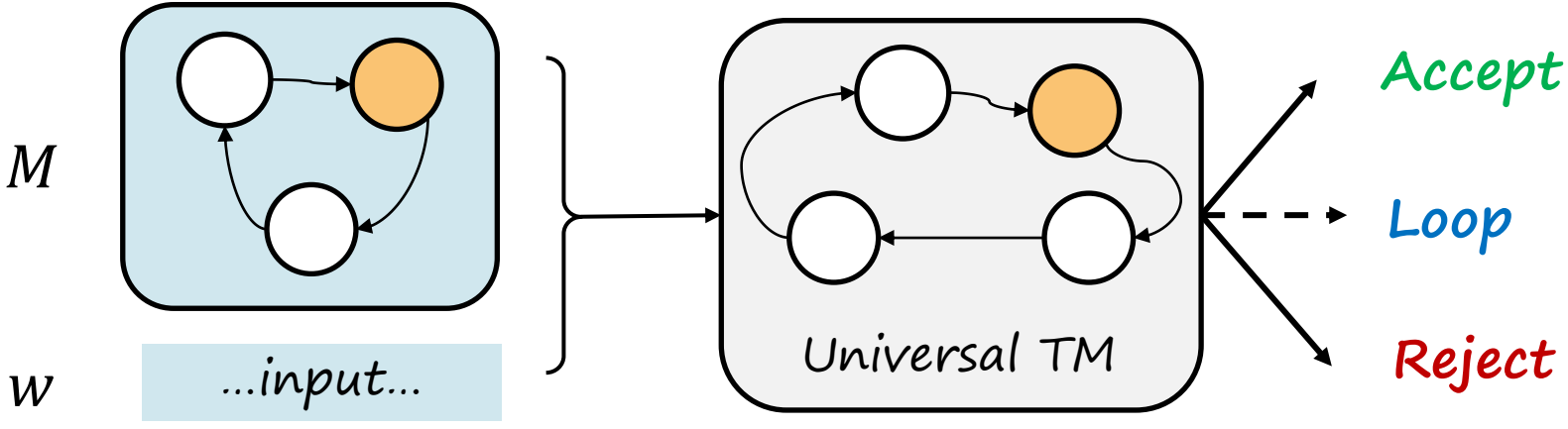
- It is possible to program a TM simulator on an unbounded-memory computer.
- We could imagine it as a method

  boolean simulateTM(TM M, string w)

- with the following behavior:
  - If M accepts w, then *simulateTM(M, w)* returns true.
  - If M rejects w, then *simulateTM(M, w)* returns false.
  - If M loops on w, then *simulateTM(M, w)* loops infinitely.

A TM Simulator

- Anything that can be done with an unbounded-memory computer can be done with a TM
- This means that there must be some TM that has the behavior of this *simulateTM* method.



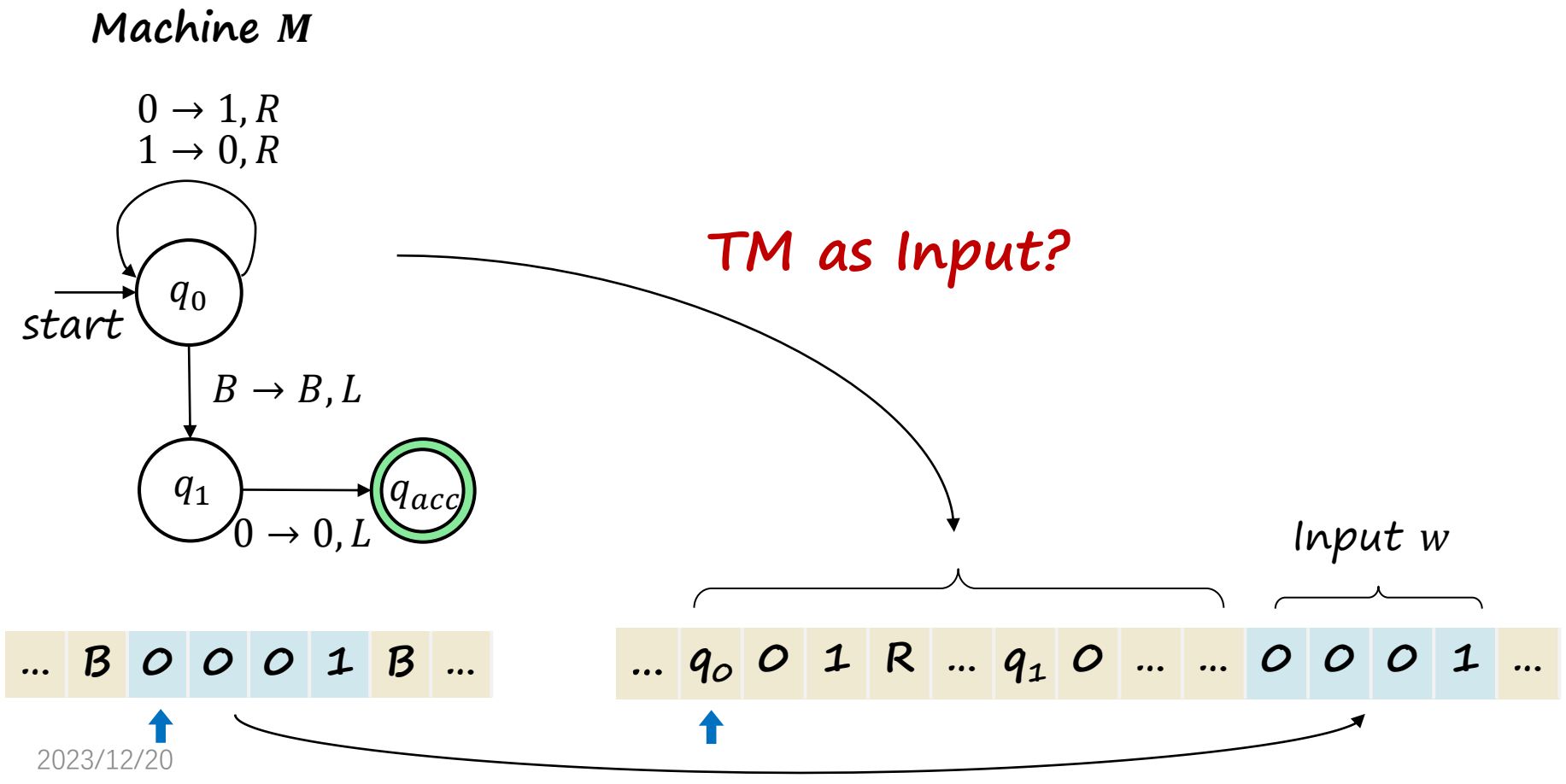
The Universal Turing Machine

- **Theorem (Turing, 1936):** There is a Turing machine U_{TM} called **the Universal Turing Machine** that, when run on an input of the form $\langle M, w \rangle$, where M is a Turing machine and w is a string, simulates M running on w and **does whatever** M does on w (accepts, rejects, or loops).
- U_{TM} behaves as follows:
  - If M accepts w, then U_{TM} accepts $\langle M, w \rangle$
  - If M rejects w, then U_{TM} rejects $\langle M, w \rangle$
  - If M loops on w, then U_{TM} loops on $\langle M, w \rangle$

Encoding Input with binary

- Input string may contains any possible character in the input alphabet of M
- But we know everything on your computer is a string over $\{0, 1\}$
- We can let the input alphabet to be $\{0,1\}$
- It not necessary to limit the alphabet as $\{0,1\}$, but only for simplicity.

The Universal Turing Machine



Encoding TM

- In order to take a Turing Machine as an input, we need to encoding the TM. Similarly, we shall encode TM with binary.
- We first assign integers to the states, tape symbols and directions
  - We assume the states are $q_1, q_2, \ldots, q_k$ for some k. $\boldsymbol{q_1}$ is the **input state** and $\boldsymbol{q_2}$ is the **only accept state**.(Is it right?)
  - The tape symbols are $X_1, X_2, \ldots, X_m$ for some m. $\boldsymbol{X_1, X_2, X_3}$ **are** $\boldsymbol{0, 1, B}$ respectively.
  - Refer to direction $\boldsymbol{L}$ **as** $\boldsymbol{D_1}$, $\boldsymbol{R}$ **as** $\boldsymbol{D_2}$

Encoding TM

- After assigning each state, symbol and direction an integer, we can encode the transition function $\boldsymbol{\delta}$.
- Suppose one transition rule is $\boldsymbol{\delta(q_i, X_j) = (q_k, X_l, D_m)}$, we shall code this rule by the string $\boldsymbol{0^i 1 0^j 1 0^k 1 0^l 1 0^m}$.
  - Notice $\boldsymbol{i, j, k, l, m}$ are at least one, so there're no occurrences of two or more consecutive 1's with in the code for a transition
- So let $\boldsymbol{C_i}$ donate the code for the ith transition rule, we can encode the whole TM as:
  - $C_1 11 C_2 11 C_3 11 \ldots 11 C_n$

Example of Code for TM

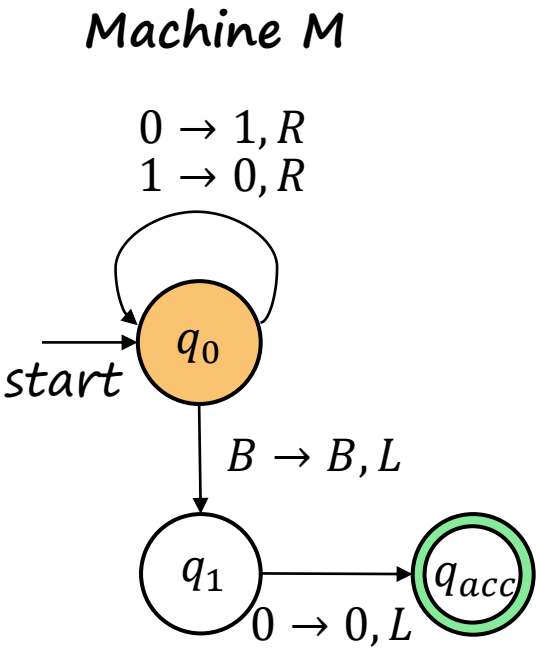
- Let a TM: $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \{\delta\}, q_1, B, q_2)$
  - $X_1 = 0, X_2 = 1, X_3 = B, D_1 = L, D_2 = R$
- Transition Function δ:
  - $\boldsymbol{\delta(q_1, 1) = (q_3, 0, R)}$
  - $\boldsymbol{\delta(q_3, 0) = (q_1, 1, R)}$
  - $\boldsymbol{\delta(q_3, 1) = (q_2, 0, R)}$
  - $\boldsymbol{\delta(q_3, B) = (q_3, 1, L)}$

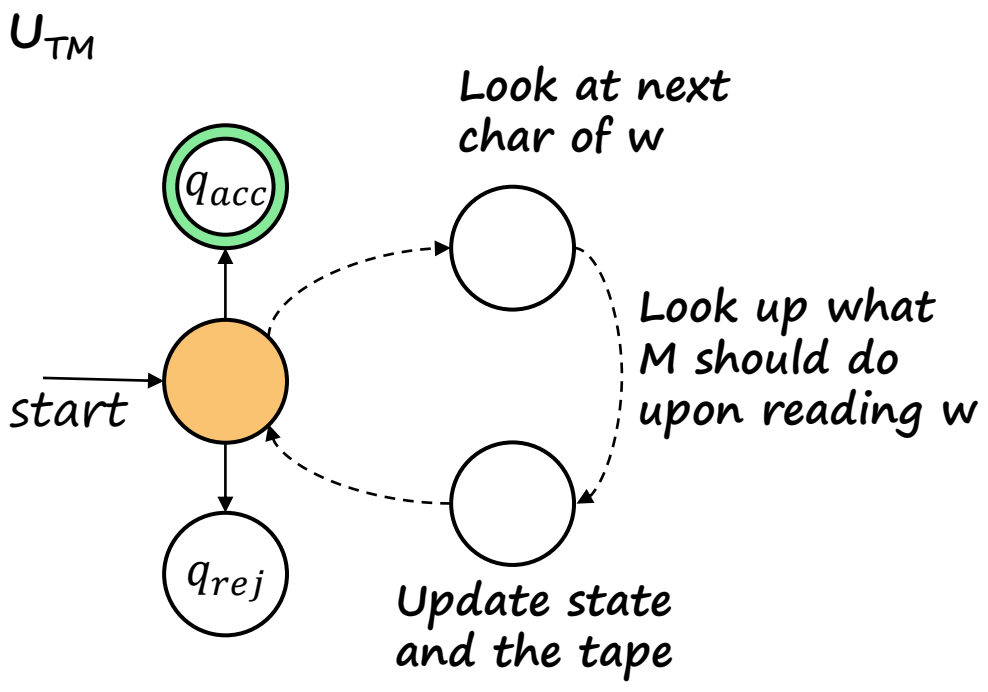
Example of Code for TM

- Let a TM: $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1, B\}, \{\delta\}, q_1, B, q_2)$
  - $X_1 = 0, X_2 = 1, X_3 = B, D_1 = L, D_2 = R$
- Transition Function δ:
  - $\boldsymbol{\delta(q_1, 1) = (q_3, 0, R) \Rightarrow \delta(q_1, X_2) = (q_3, X_1, D_2) \Rightarrow 01001000101 00}$
  - $\boldsymbol{\delta(q_3, 0) = (q_1, 1, R) \Rightarrow \delta(q_3, X_1) = (q_1, X_2, D_2) \Rightarrow 00010101001 00}$
  - $\boldsymbol{\delta(q_3, 1) = (q_2, 0, R) \Rightarrow \delta(q_3, X_2) = (q_2, X_1, D_2) \Rightarrow 00010010101 00}$
  - $\boldsymbol{\delta(q_3, B) = (q_3, 1, L) \Rightarrow \delta(q_3, X_3) = (q_3, X_2, D_1) \Rightarrow 00010001000100 10}$
- **Code for this TM:**

0100100010100110001010100100110001001001010011000100010001001 0

The Universal Turing Machine

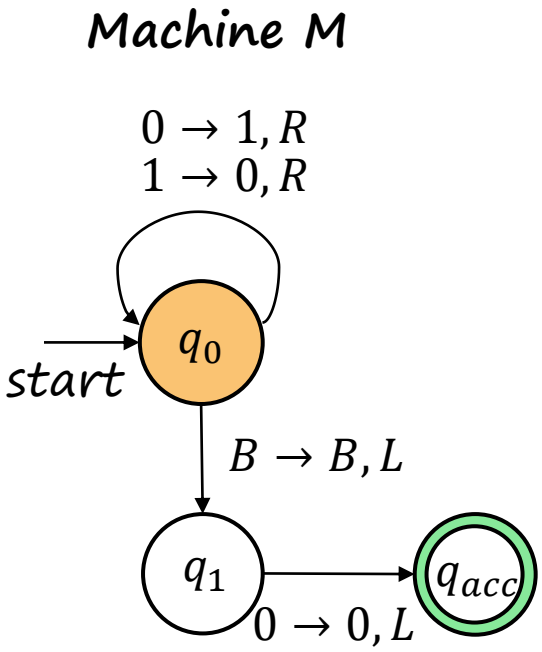


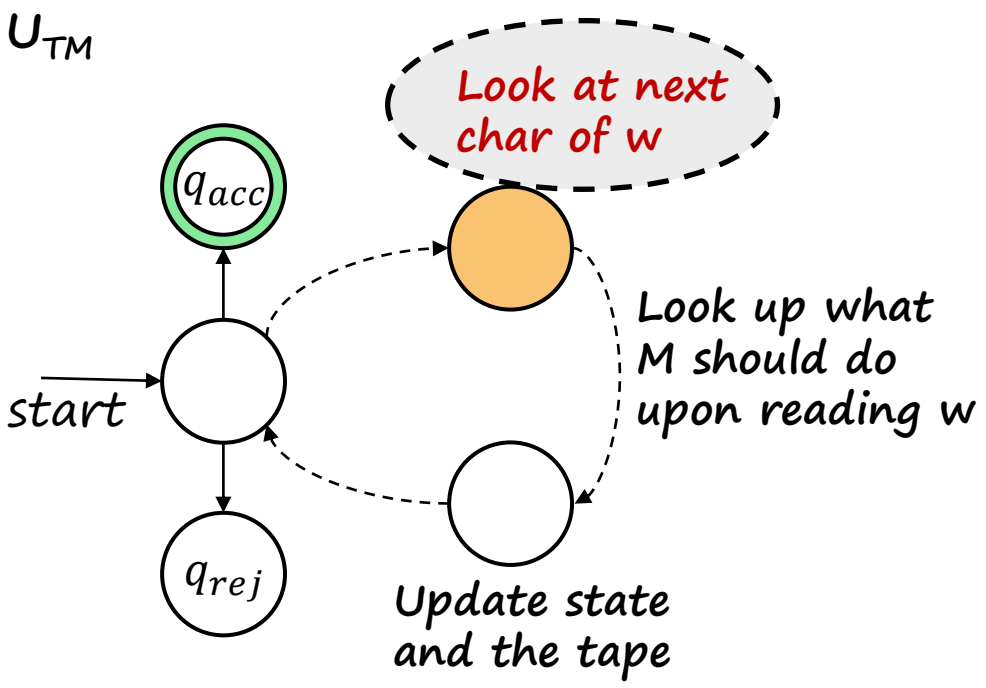


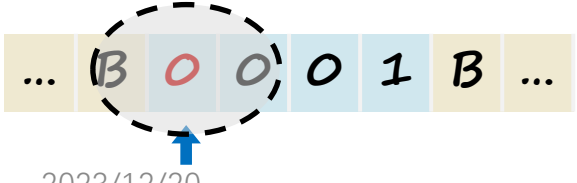
...	B	0	0	0	1	B	...

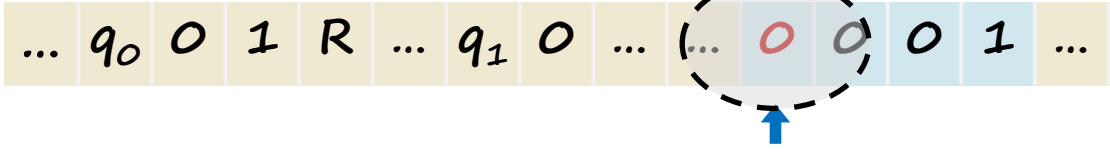
...	q_0	0	1	R	...	q_1	0	...	...	0	0	0	1	...

The Universal Turing Machine

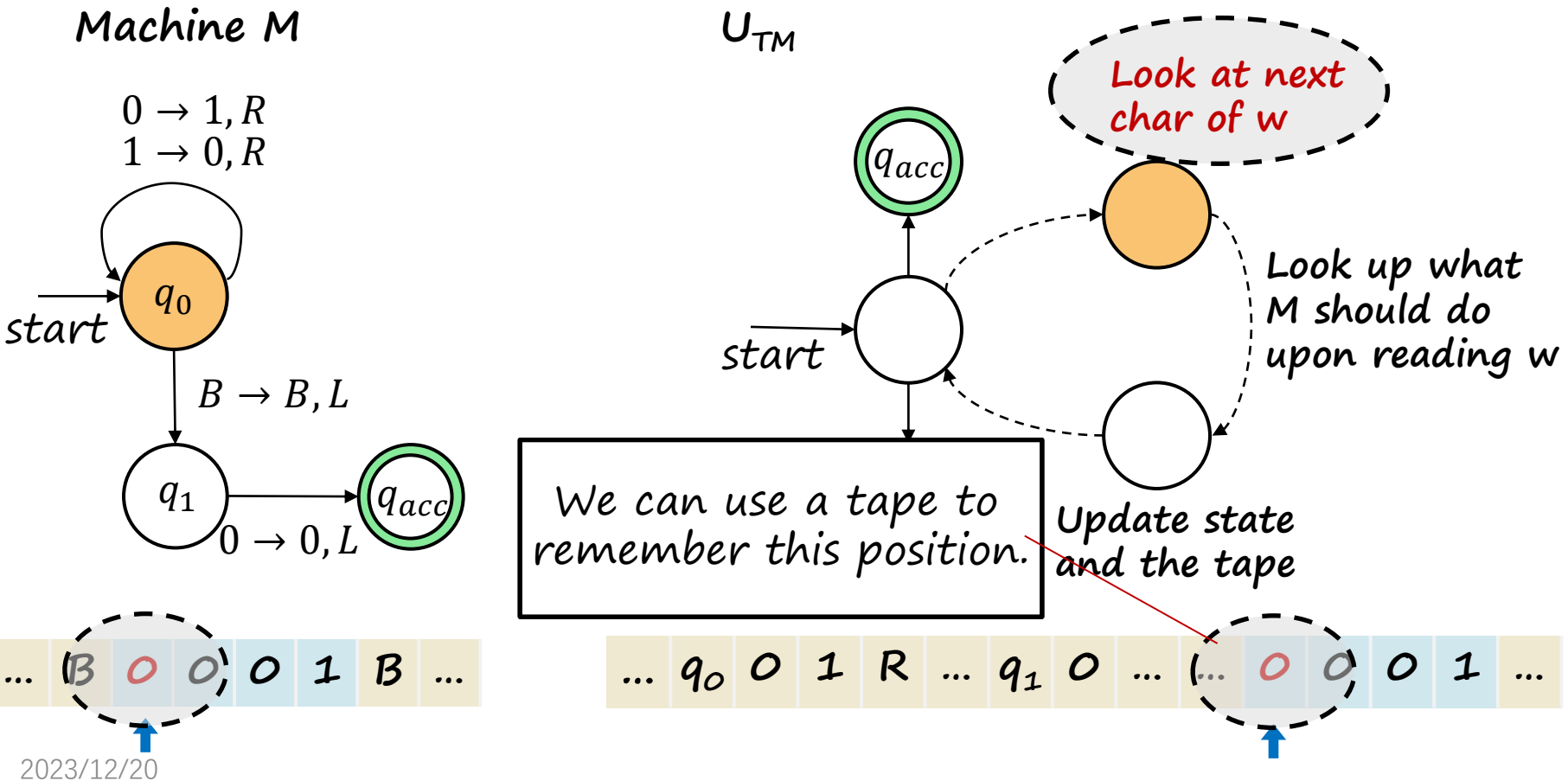




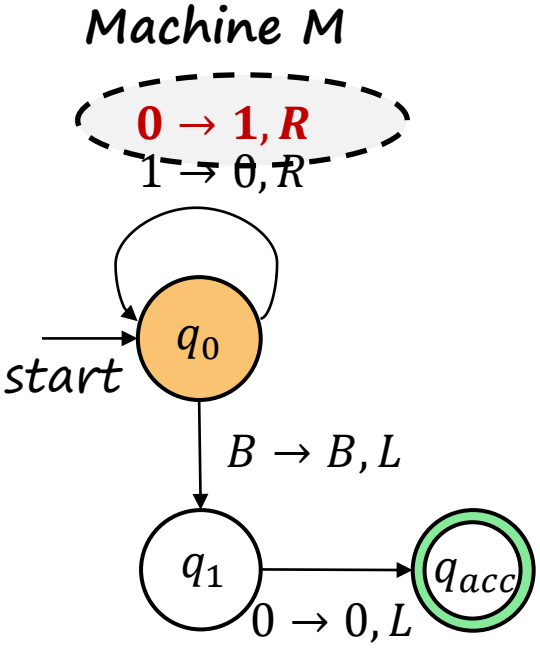


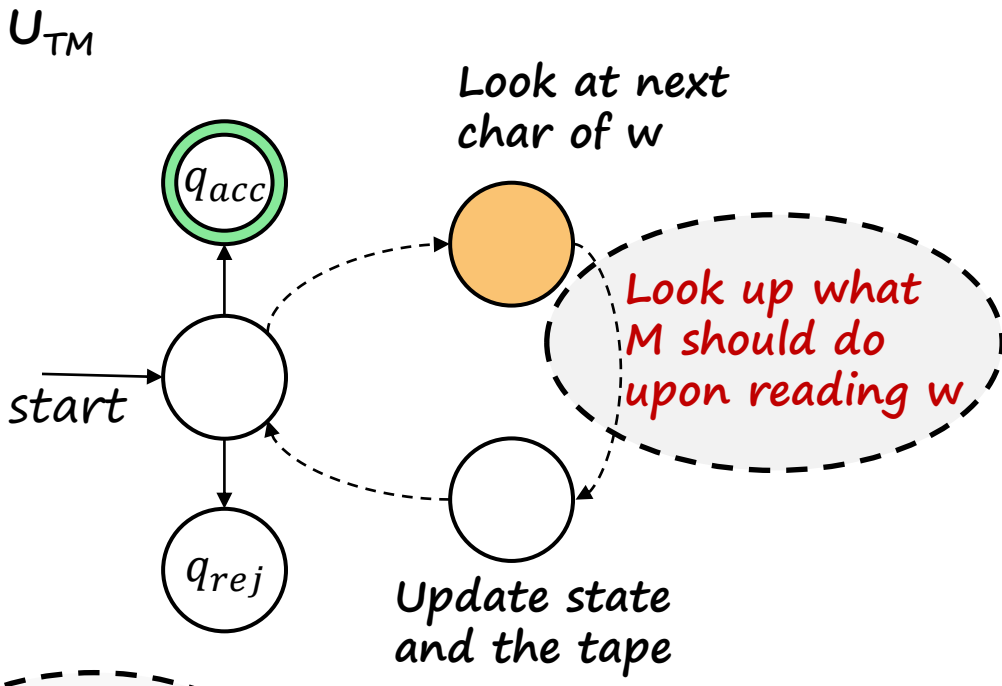


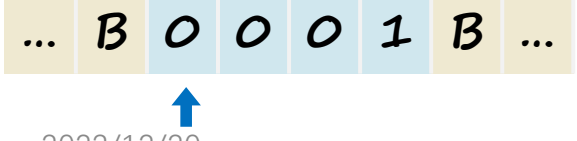
The Universal Turing Machine

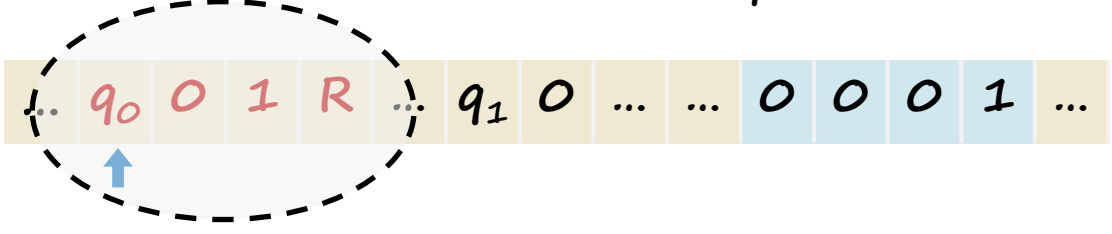


The Universal Turing Machine

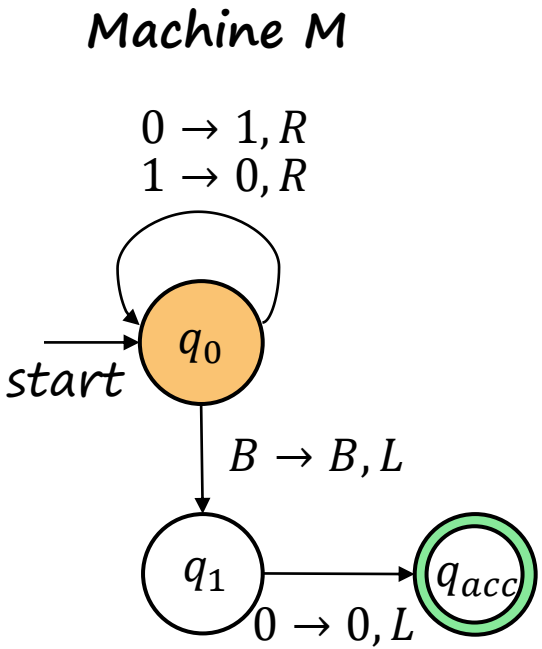


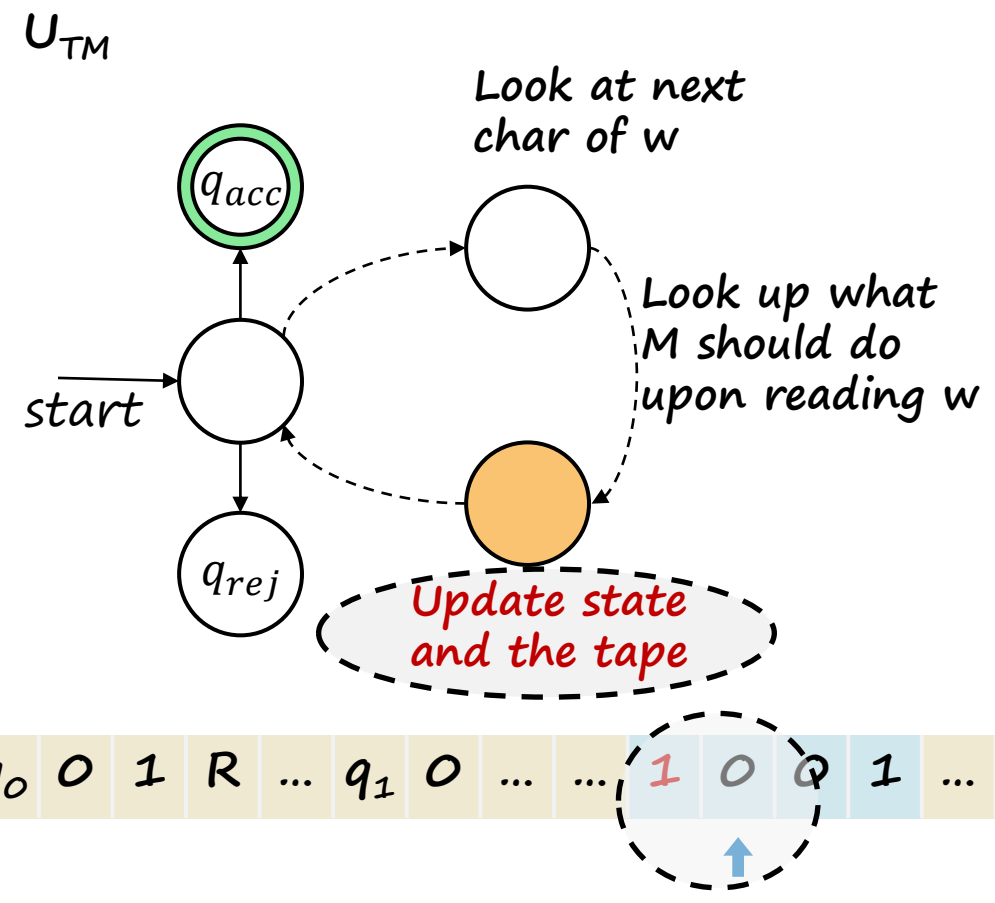




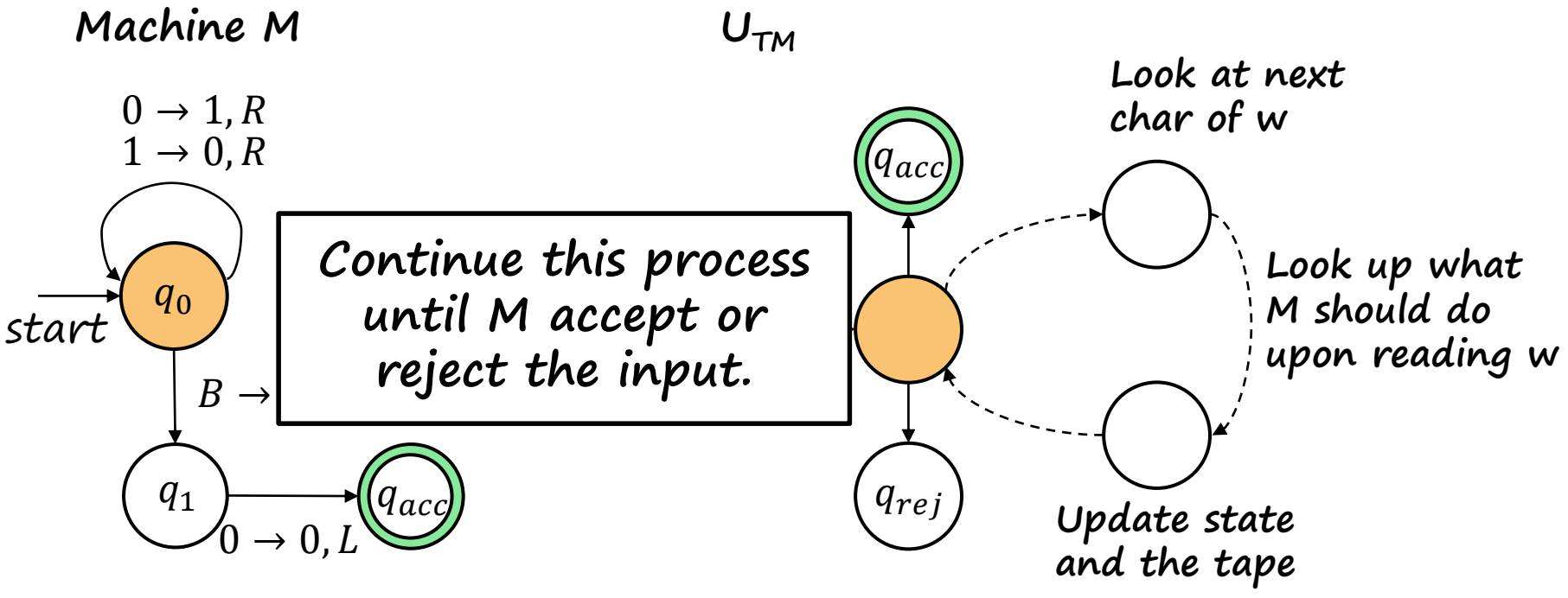


The Universal Turing Machine





The Universal Turing Machine



…	B	1	0	0	1	B	…

…	q_0	0	1	R	…	q_1	0	…	…	1	0	0	1	…

The Language of U_{TM}

- Recall that the language of a TM is the set of all strings that TM accepts.
- $\boldsymbol{U_{TM}}$ when run on a string $\langle \boldsymbol{M}, \boldsymbol{w} \rangle$, where $\boldsymbol{M}$ is a TM and $\boldsymbol{w}$ is a string, will
  - Accept $\langle M, w \rangle$ if M accepts w
  - Reject $\langle M, w \rangle$ if M rejects w
  - Loop on $\langle M, w \rangle$ if M loops on w.

The Language of U_{TM}

- The universal language, donated $\boldsymbol{L_u}$, is the language of the U_{TM}

$$L_u = L(U_{TM}) = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w\}$$

$$= \{\langle M, w\rangle | M \text{ is a TM and } w \in L(M)\}$$

- Useful fact:

$$\boldsymbol{\langle M, w\rangle \in L_u \Leftrightarrow M \text{ accepts } w}$$

- Because $L_u = L(U_{TM})$, we know that $L_u \in RE$

The Language of U_{TM}

- If M accepts w, then we have:
  - U_{TM} accepts $\langle M, w \rangle$
  - U_{TM} accepts $\langle U_{TM}, \langle M, w \rangle \rangle$
  - U_{TM} accepts $\left\langle U_{TM}, \langle U_{TM}, \langle M, w \rangle \rangle \right\rangle$
  - ….

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.

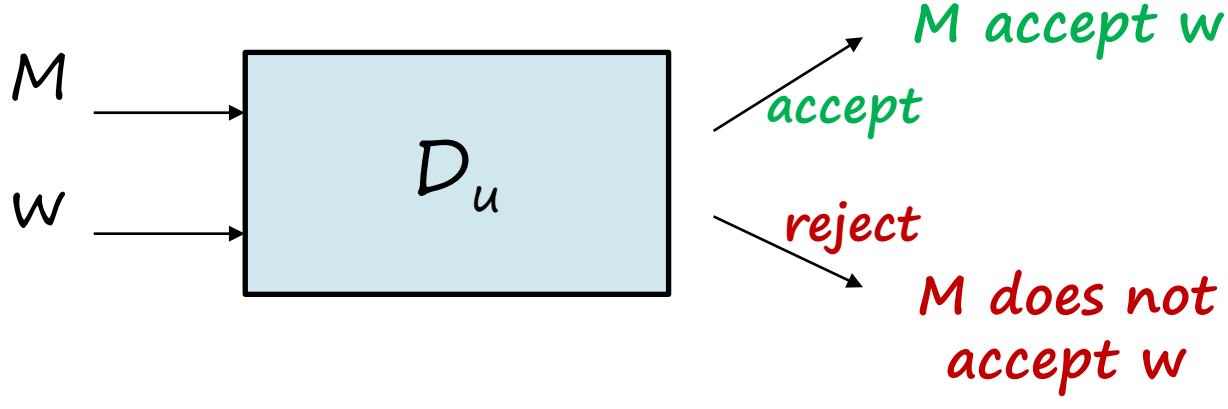
2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

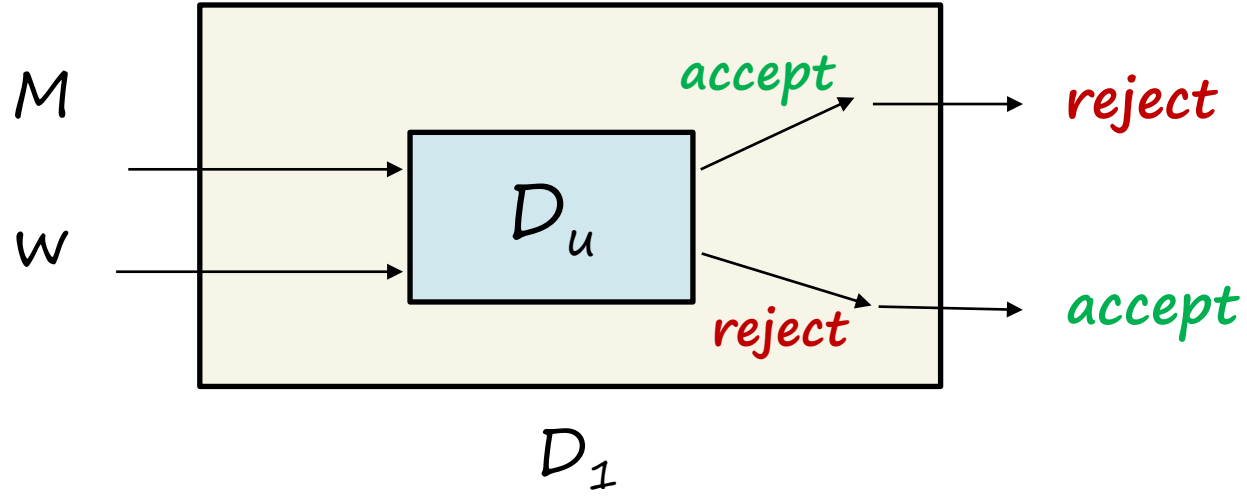
A Decider for L_u

- We know that $L_u \in \mathbf{RE}$, but whether $L_u \in \mathbf{R}$?
- That is, whether there is a decider D_u for L_u?



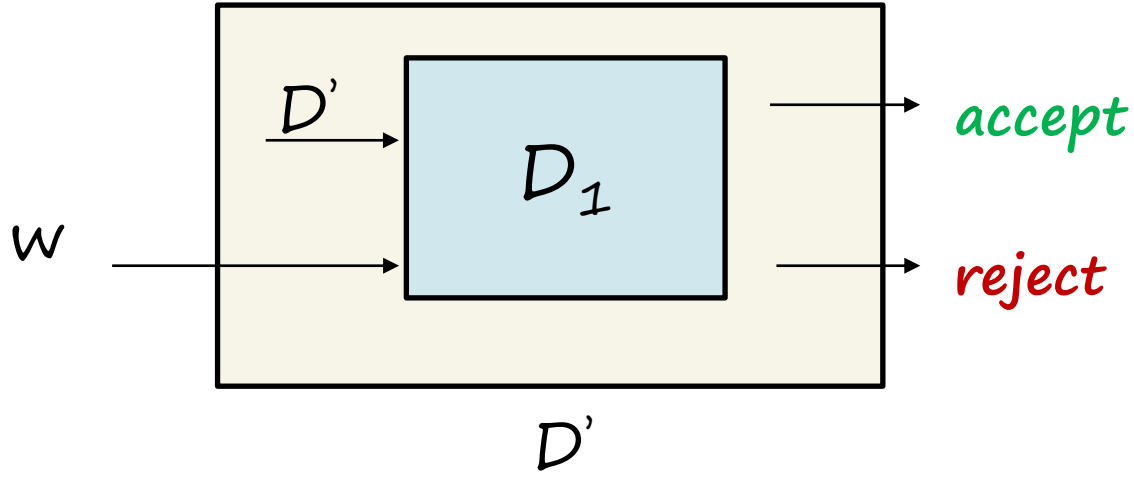
A Decider for L_u

- If D_u exists, we can build a new TM D_1 based on D_u
- D_1 takes a TM M and string w as inputs, and feeds them to D_u, then outputs the **reverse result** of D_u.



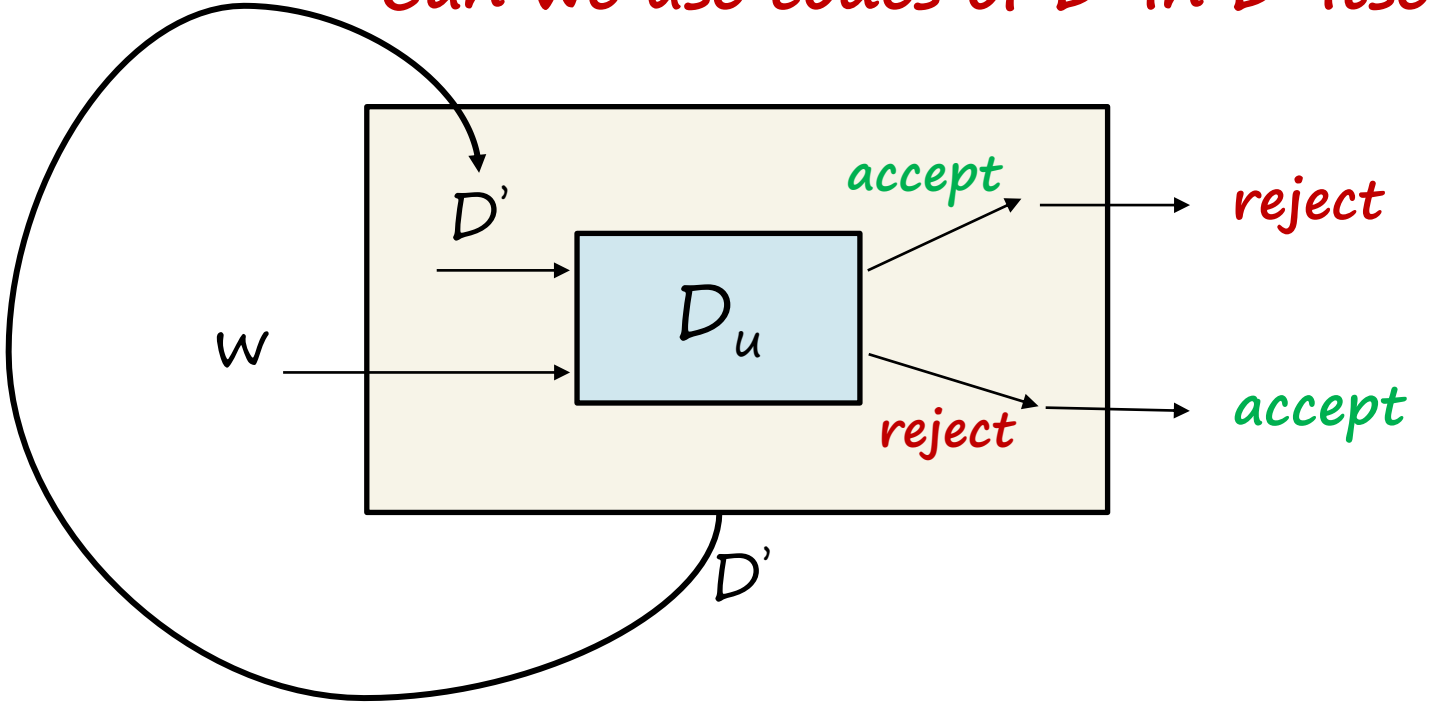
A Decider for L_u

- Then we can build another TM D' based on D_1
- D' only takes a string w as input, and feeds **its own code** and w to D_1, then outputs the result of D_1.

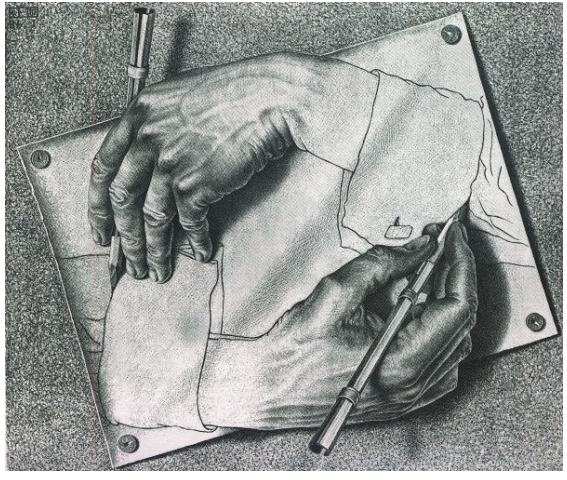


A Decider for L_u

Can we use codes of D' in D' itself?



Self-Reference



Sets that do not contain itself.

"This sentence is wrong."

Self-Referential Software

- A **Quine** is a program that, when running, prints its own source code.
- Quines aren't allowed to just read the file containing their source code and print it out; that's cheating (and technically incorrect if someone changes that file!)
- How would you write such a program?

Example: Quine

```
#include <stdio.h>
char*f="#include <stdio.h>
char*f=%c%s%c;
main(){printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

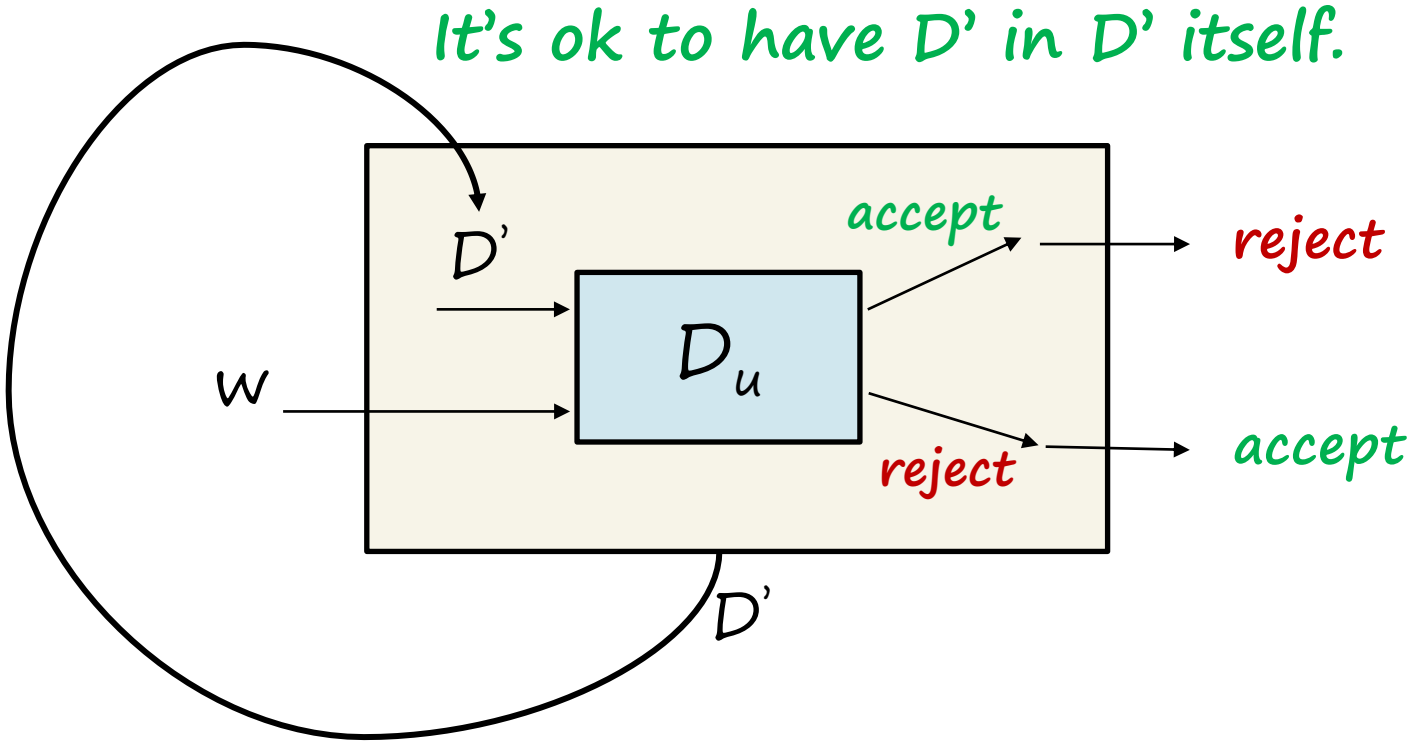
http://www.nyx.net/~gthompso/quine.htm

Self-Referential Software

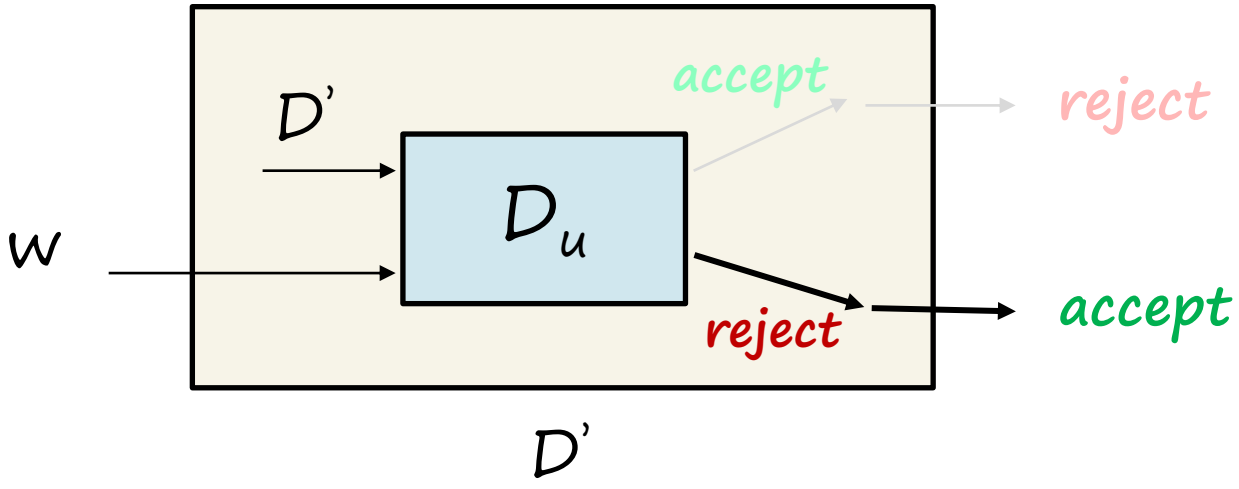
- Going forward, assume that any program can be augmented to include a method called *mySource()* that returns a string representation of its source code.
- Then we can take the whole program codes as an input though we don't know what the whole program codes are, and even what we write now will change the final code itself.

```
char * mySource(){…}
int main(){
  char * s = mySource();
  ......
}
```

A Decider for L_u

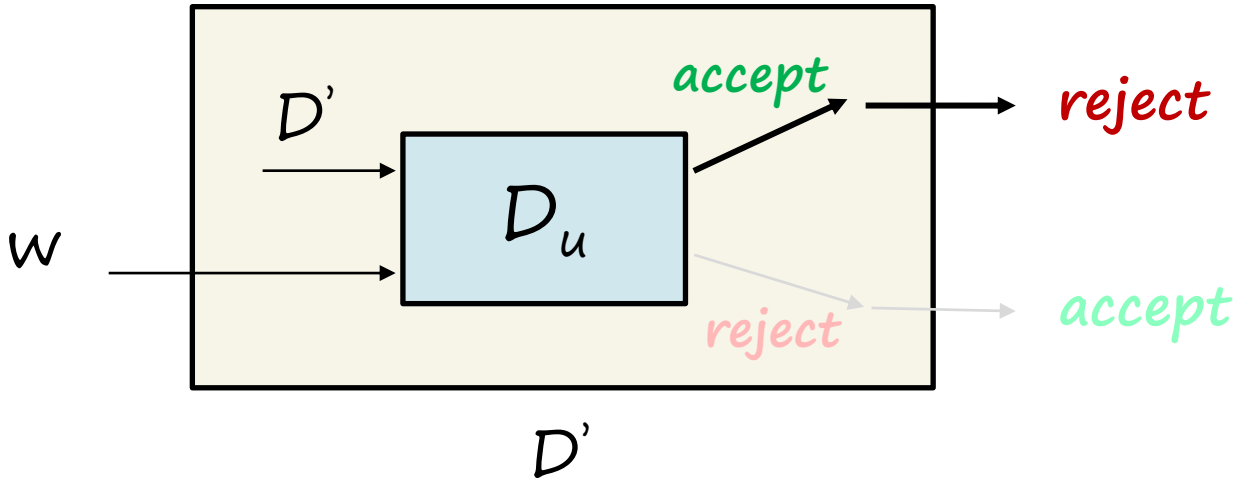


A Decider for L_u



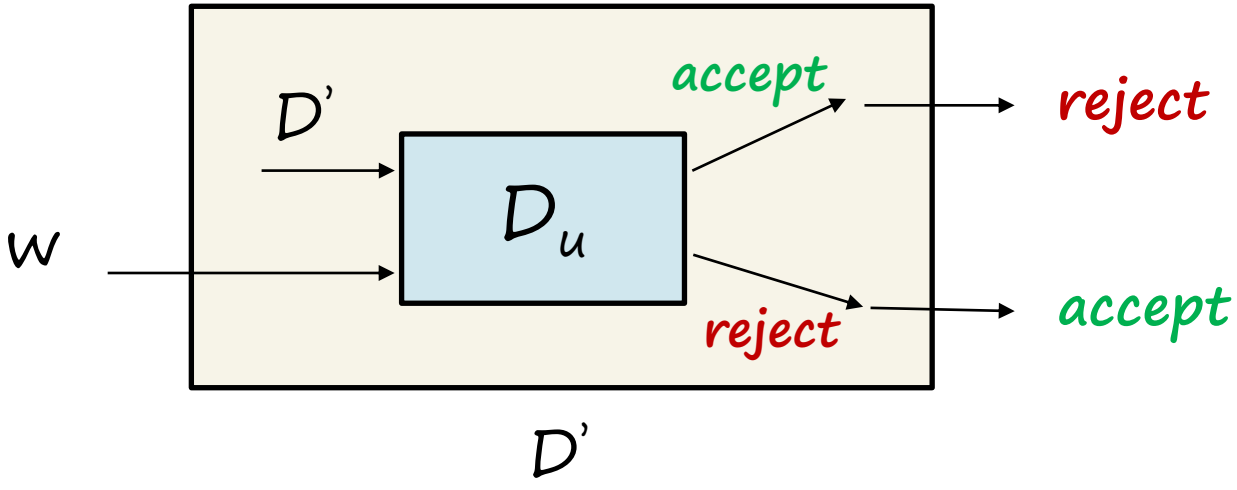
- If D' accepts w.
  - Then D_u rejects $\langle D', w \rangle$, which means that D' rejects w

A Decider for L_u



- If D' accepts w.
  - Then D_u rejects $\langle D', w \rangle$, which means that D' rejects w
- If D' rejects w.
  - Then D_u accepts $\langle D', w \rangle$, which means that D' accepts w

A Decider for L_u



$\boldsymbol{D'}$ accepts $\boldsymbol{w} \Leftrightarrow \boldsymbol{D'}$ rejects $\boldsymbol{w}$

D_u can not exist!

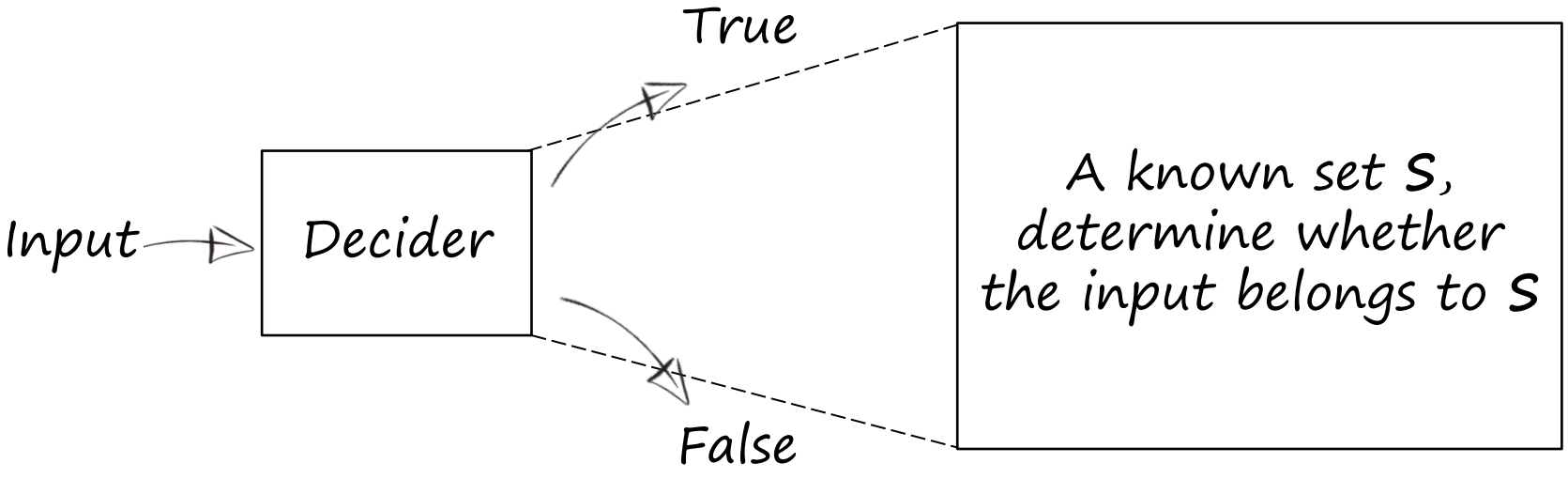
$L_u \notin \mathbf{R}$

- As there are no deciders for L_u, L_u is not in the class of **R.**
- **Undecidable(不可判定):** there is **no algorithm** that can determine whether a program will definitely accept or reject an input.
- Intuition: The only general way to find out what a program will do is to run it.

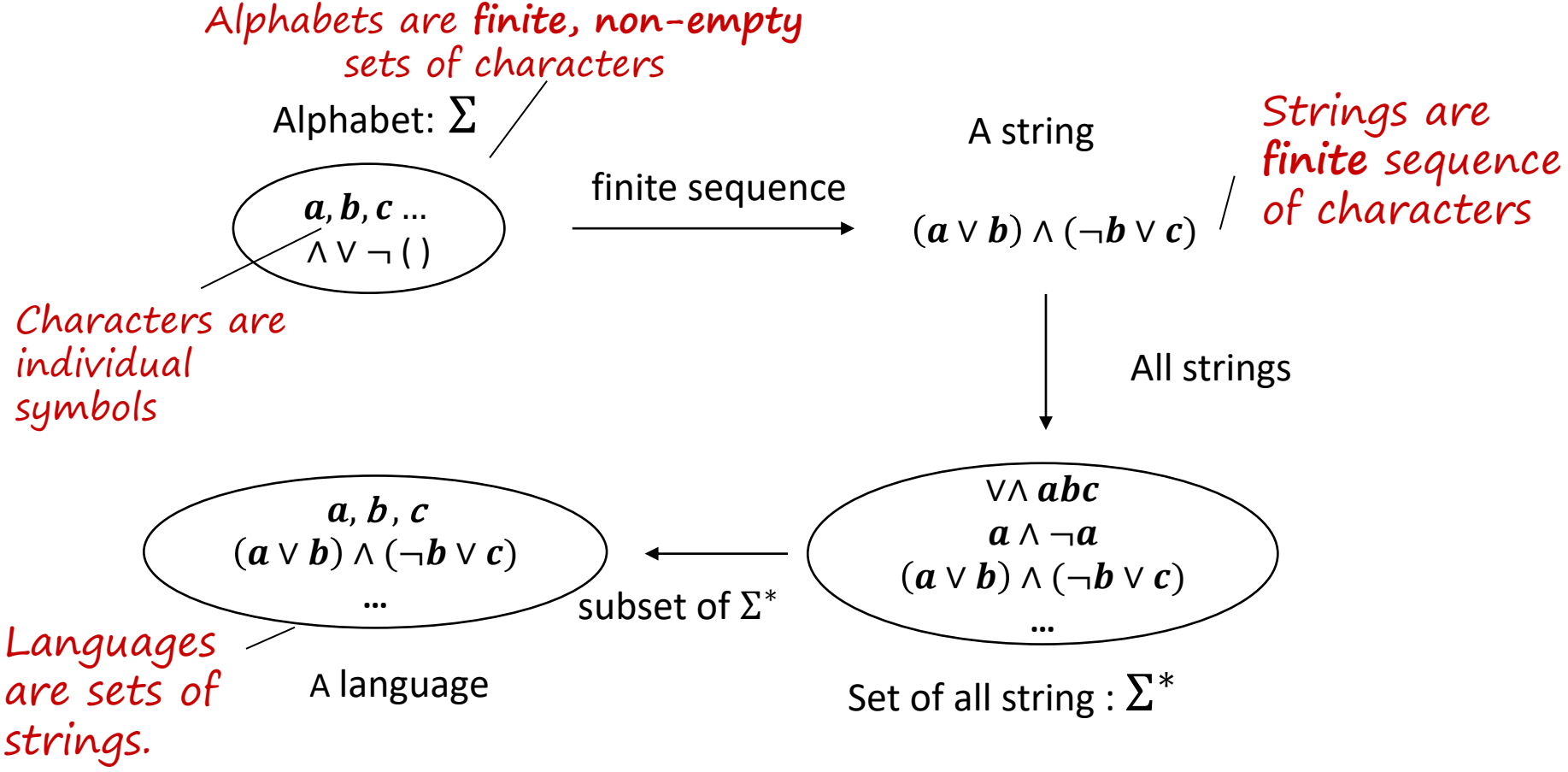
Computable（可计算的）

- **A computational problem is computable if there is a procedure (or algorithm) for it which:**
  - Rigorously follows some finite instructions, requires no ingenuity.
  - Always finishes (terminates) after a finite number of steps.

Solve Decision Problem



Formal Language Theory

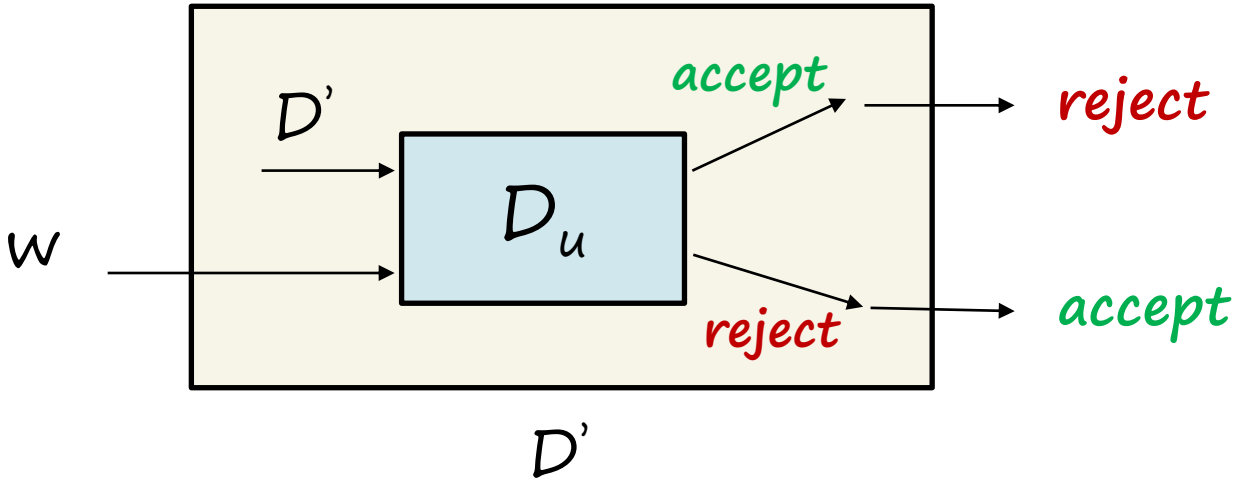


What problems can a computer solve?

What languages can a computer recognize?

What languages can TM recognize?

An Undecidable Problem



$\boldsymbol{D'}$ accepts $\boldsymbol{w} \Leftrightarrow \boldsymbol{D'}$ rejects $\boldsymbol{w}$

D_u can not exist!

$\mathbf{R} \neq \mathbf{RE}$

- We found a language $L_u \in \mathbf{RE}$, but $L_u \notin \mathbf{R}$, which means $\mathbf{R} \neq \mathbf{RE}$
- Because R ≠ RE, there is a difference between decidability and recognizability: