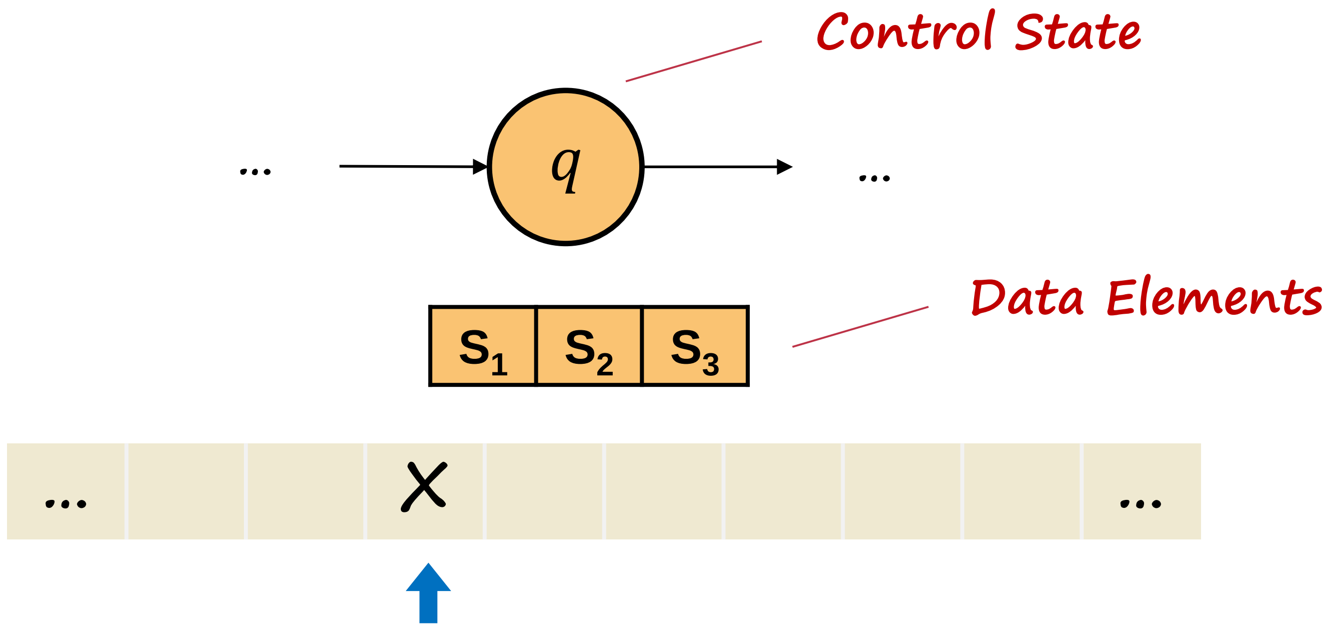
Turing Machine III

Zeyu Mi

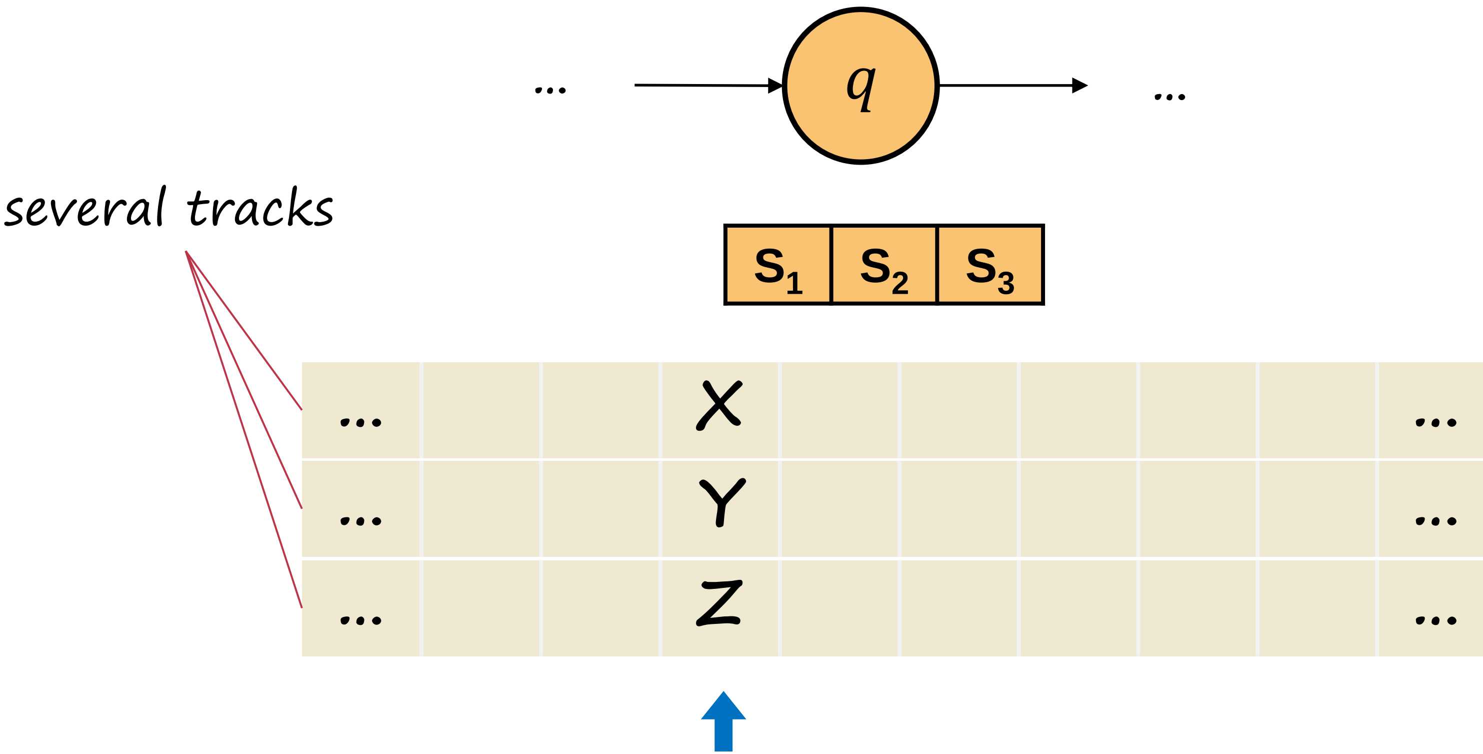
2019-12-9

Adapted From:
IALC 9.1.2, 9.2.1, 9.2.2, 9.2.3
Stanford CS103 Turing Machine

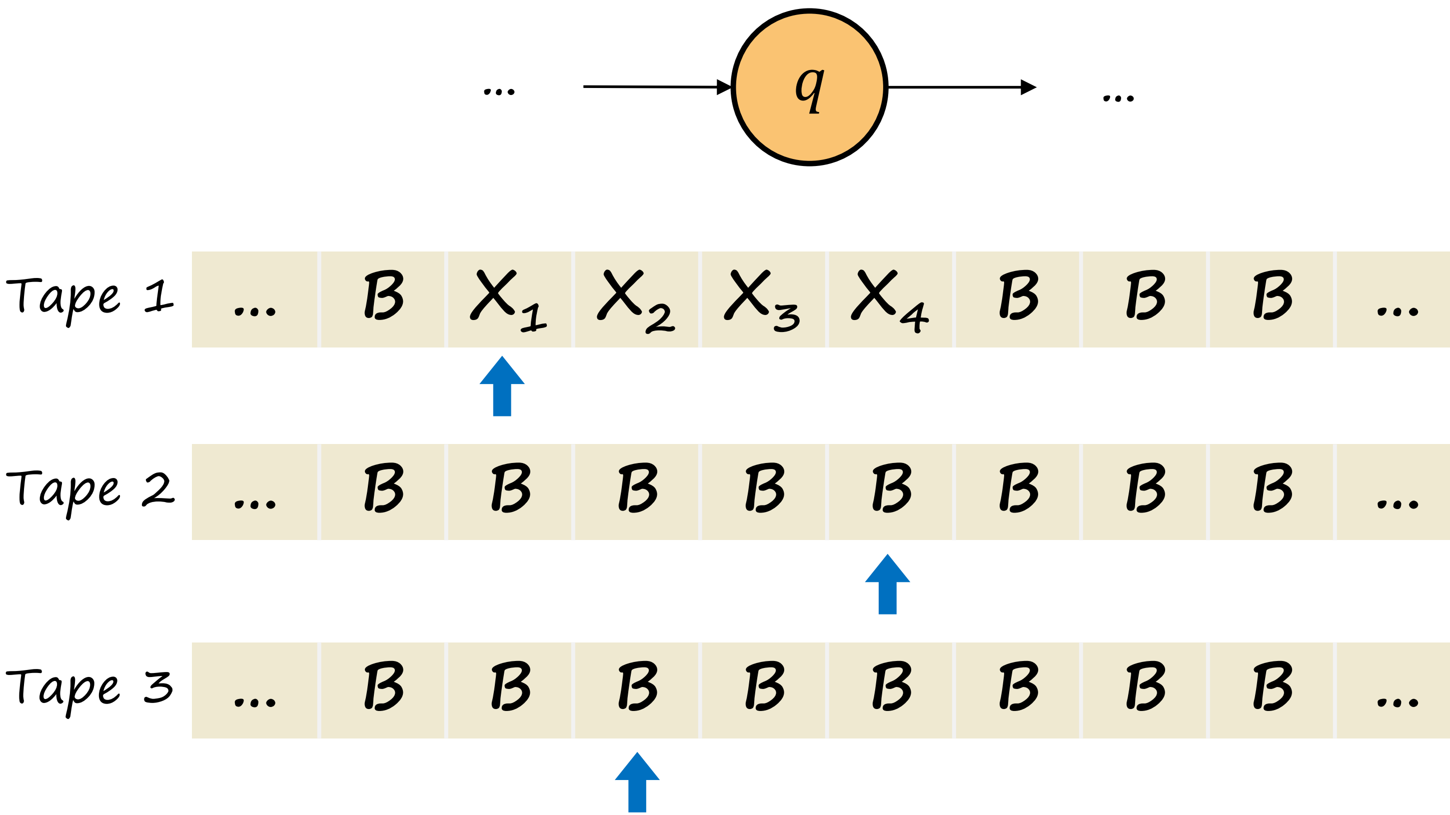
Review: TM with Storage



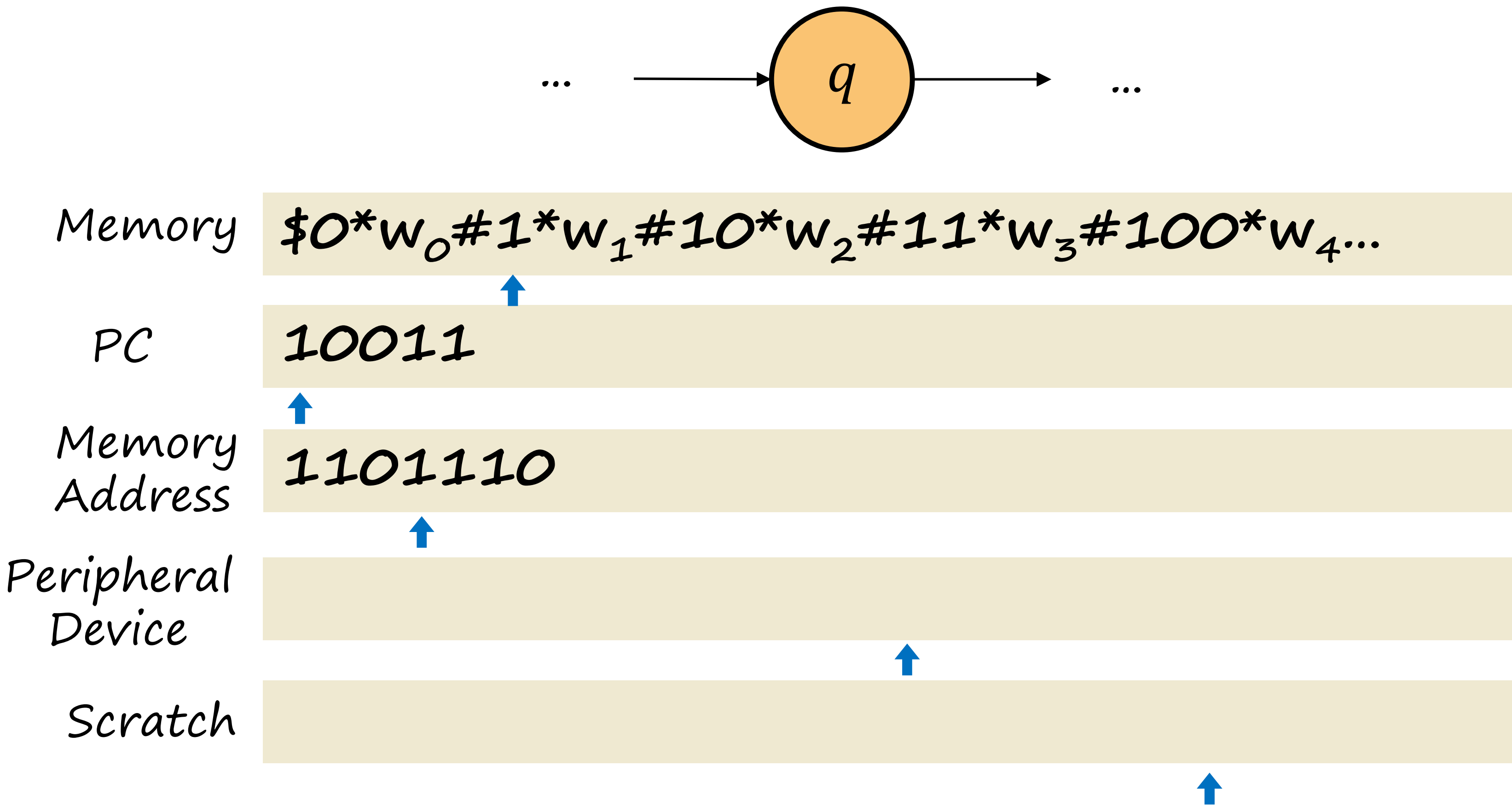
Review: Multitrack TM



Review: Multi-tape TM



Review: TM ≈ Idealized Computer



Effective Computation

- An ***effective method of computation*** is a form of computation with the following properties:
  - The computation consists of a set of steps.
  - There are fixed rules governing how one step leads to the next.
  - Any computation that yields an answer does so in finitely many steps.
  - Any computation that yields an answer always yields the correct answer.
- This is not a formal definition. Rather, it's a set of properties we expect out of a computational system.

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”.

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.

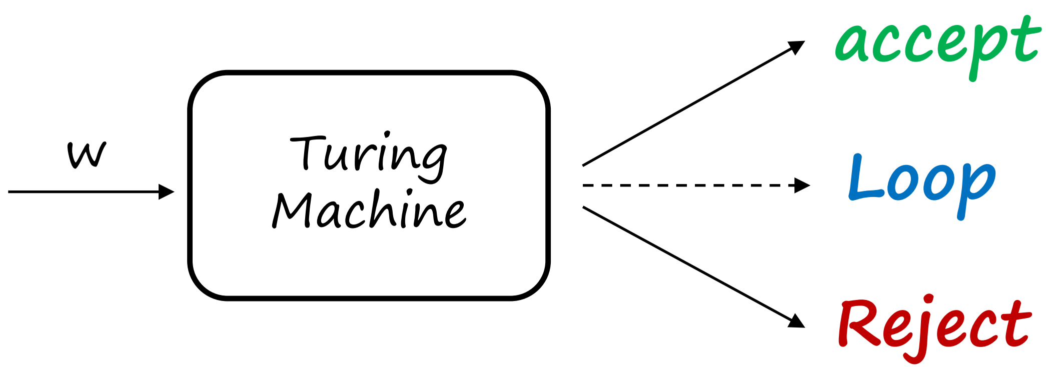
2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

Part2. Undecidability.

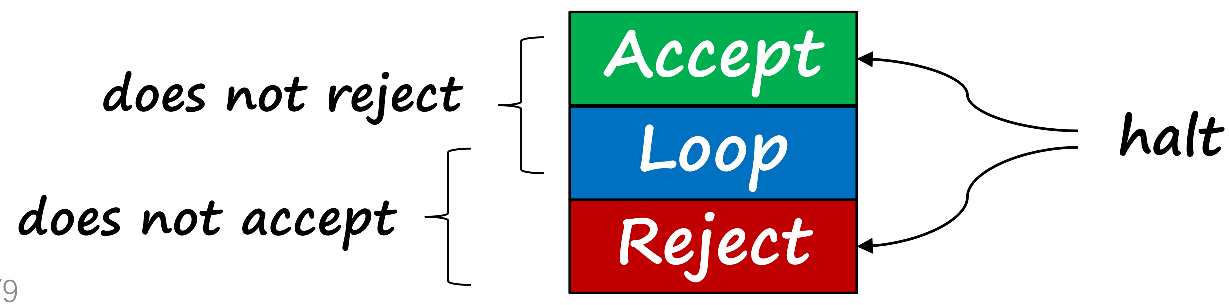
Output of TM

- 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

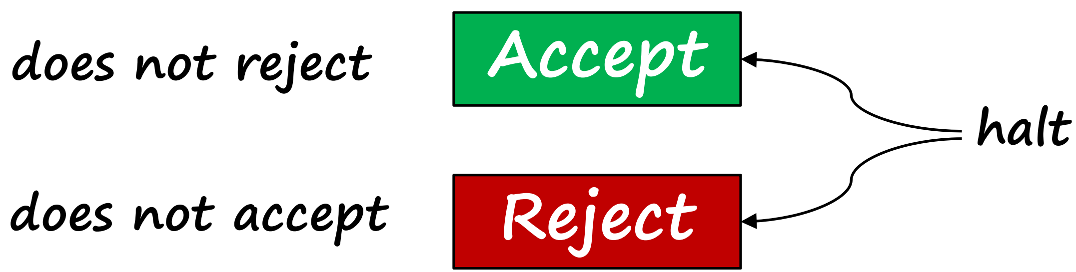
- We've known that for a certain language L, if there exists a TM $\boldsymbol{M}$ such that $L = L(\boldsymbol{M})$, then we call L is a **recursively enumerable language(递归可枚举语言)**, or ***RE***.
- So for any RE $\boldsymbol{L}$, a TM $\boldsymbol{M}$ that $L(M) = L$, and any string $\boldsymbol{w}$
  - 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, M will tell you correct on every correct input, but M is not necessary to tell you wrong on every wrong input.
  - You can not determine the input is correct or wrong until M halts.
- But a problem is solvable(computable) if the computation process finishes in finite steps.

Decider

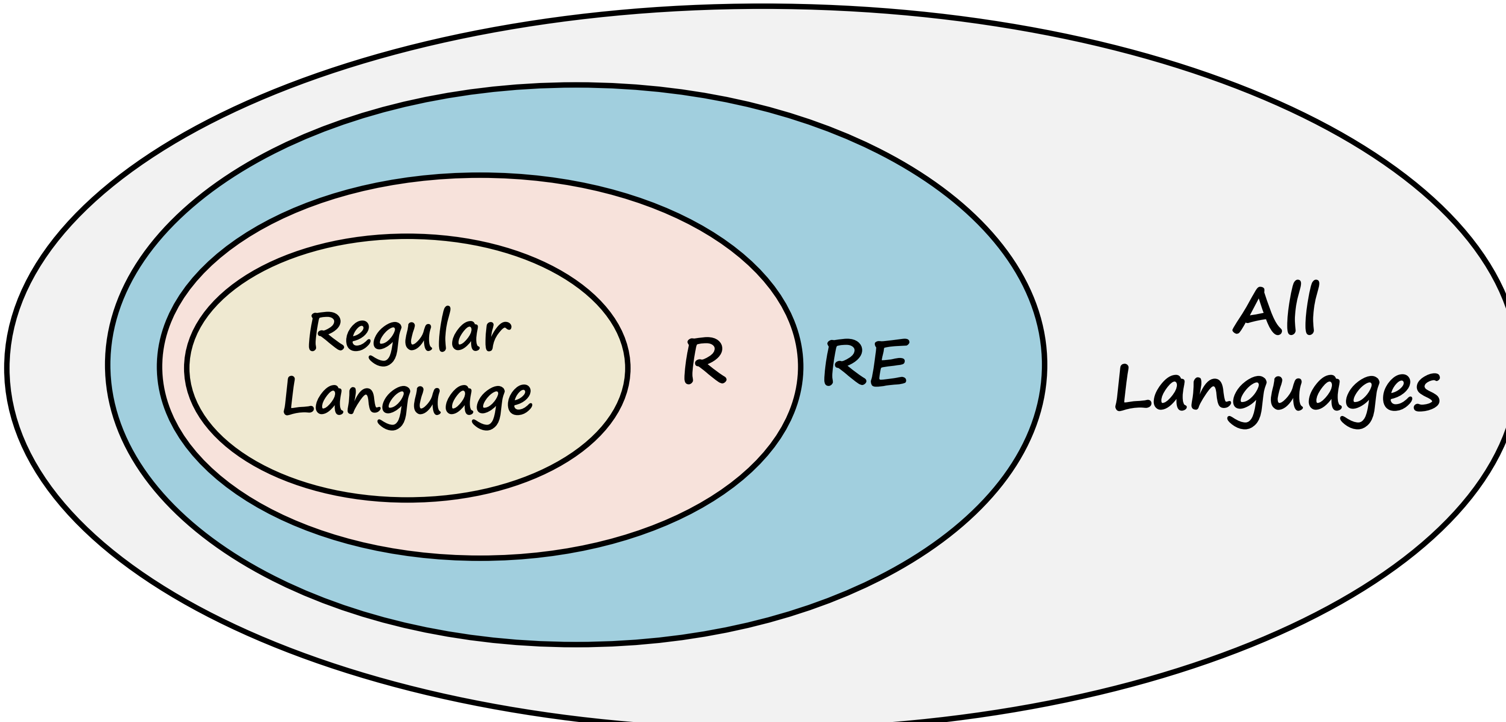
- Some Turing machines always halt; they never go into an infinite loop.
- If $\boldsymbol{M}$ is a TM and $\boldsymbol{M}$ halts on every possible input, then we say that $\boldsymbol{M}$ is a decider.
- For deciders, accepting is the same as not rejecting and rejecting is the same as not accepting.



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

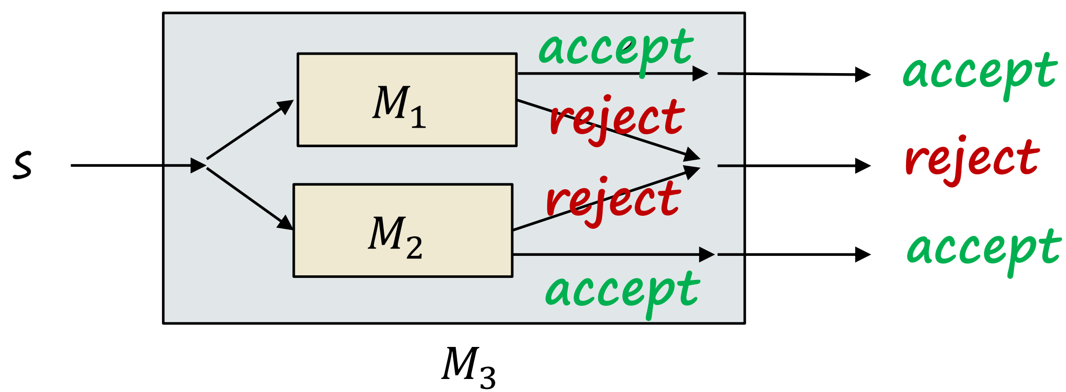


Properties of R & RE

- Union/Intersection of two **RE** language is **RE**.
- Union/Intersection of two **R** language is **R**.

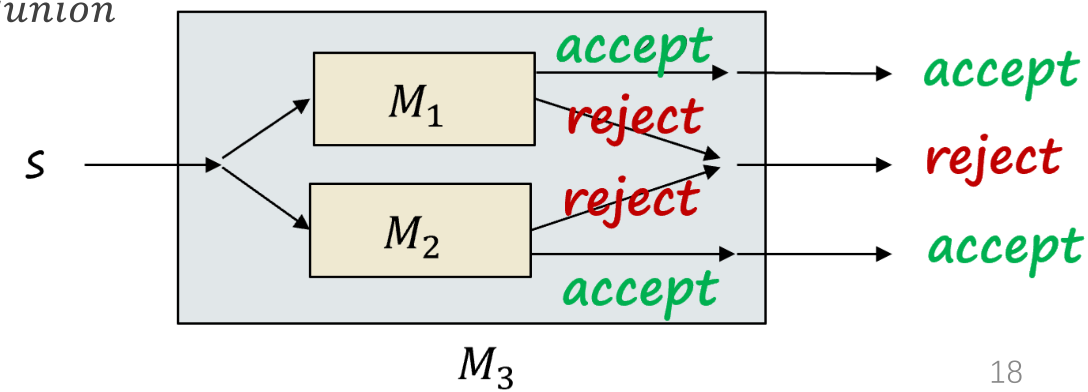
Union of Two RE

- Let L_1, L_2 be two RE language, $L_{union} = L_1 \cup L_2$
- According to definition, there are two TMs M_1, M_2 accept L_1, L_2
- We can design a new TM M_3 based on M_1 and M_2
- M_3 concurrently simulates M_1 and M_2, if any one accepts, then M_3 accepts. (How to simulate this process?)



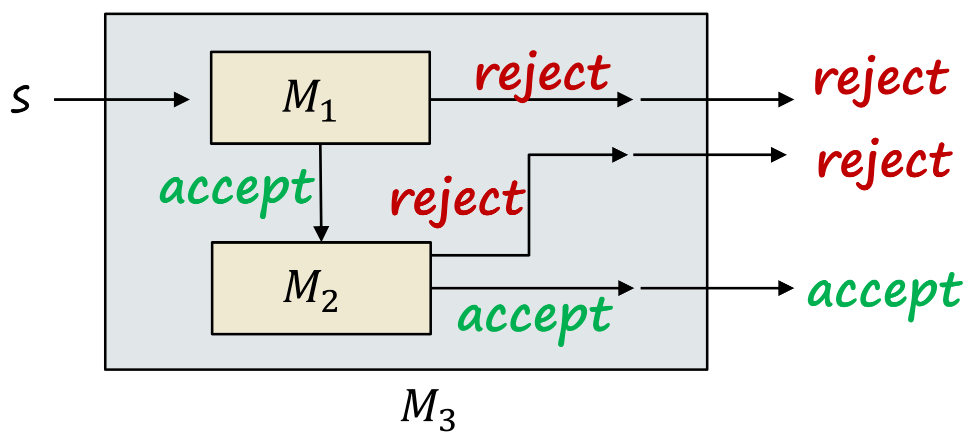
Union of Two RE

- If $s \in L_{union}$
  $\Rightarrow s \in L_1 \vee s \in L_2$
  $\Rightarrow M_1 \; or \; M_2 \; accepts \; s \Rightarrow M_3 \; accepts \; s$
- If $s \notin L_{union}$
  $\Rightarrow s \notin L_1 \wedge s \notin L_2$
  $\Rightarrow M_1 \; and \; M_2 \; do \; not \; accept \; s \Rightarrow M_3 \; does \; not \; accept \; s$
- M_3 is a TM recognizing L_{union}
- L_{union} is RE



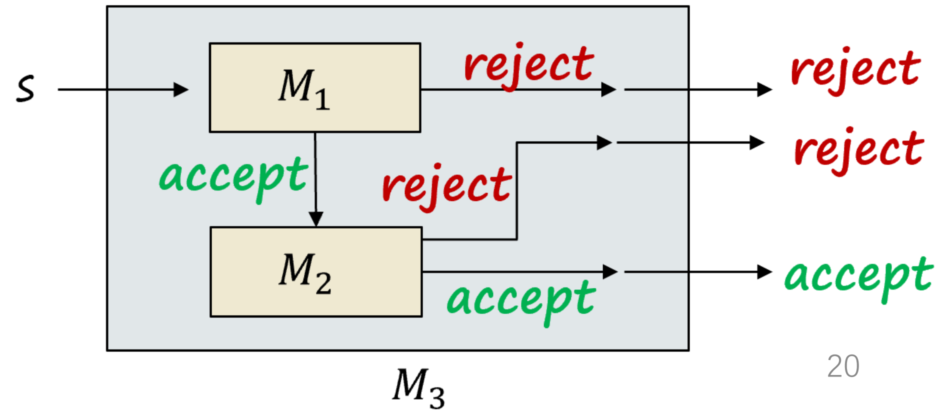
Intersection of Two RE

- Let L_1, L_2 be two RE language, $L_{intersection} = L_1 \cap L_2$
- According to definition, there are two TMs M_1, M_2 accept L_1, L_2
- We can design a new TM M_3 based on M_1 and M_2
- M_3 first simulates M_1, if M_1 accepts then simulates M_2. If M_2 still accepts, then M_3 accepts. (How to simulate this process?)



Intersection of Two RE

- If $s \in L_{intersection}$
  $\Rightarrow s \in L_1 \wedge \ s \in L_2$
  $\Rightarrow M_1 \ and \ M_2 \ accepts \ s \Rightarrow M_3 \ accepts \ s$
- If $s \notin L_{intersection}$
  $\Rightarrow s \notin L_1 \vee \ s \notin L_2$
  $\Rightarrow M_1 \ or \ M_2 \ does \ not \ accept \ s \Rightarrow M_3 \ does \ not \ accept \ s$
- M_3 is a TM recognizing $L_{intersection}$
- $L_{intersection}$ is RE



Properties of R & RE

- Union/Intersection of two **RE** language is **RE**.
- Union/Intersection of two **R** language is **R**.

It can be similarly proved.

Complements of Language

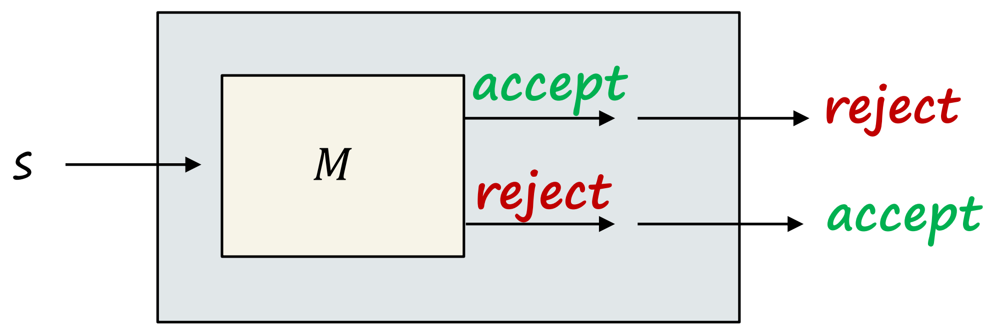
- A language $\boldsymbol{L}$'s complement $\overline{\boldsymbol{L}}$ is the set containing all valid strings not in $\boldsymbol{L}$
  - $\overline{\boldsymbol{L}} = \boldsymbol{\Sigma}^* - \boldsymbol{L}$, recall that $\boldsymbol{\Sigma}^*$ is total available string set
- Properties:
  - If $\boldsymbol{L}$ is recursive, so is $\overline{\boldsymbol{L}}$
  - If both $\boldsymbol{L}$ and $\overline{\boldsymbol{L}}$ is RE, then $\boldsymbol{L}$ is recursive

Complements of Language

If L is recursive, so is $\overline{L}$.

Proof:

- Let $L = L(M)$ for some TM M that always halts. We construct a TM $\overline{M}$ such that $\overline{L} = L(\overline{M})$ by "flipping" the result of M.

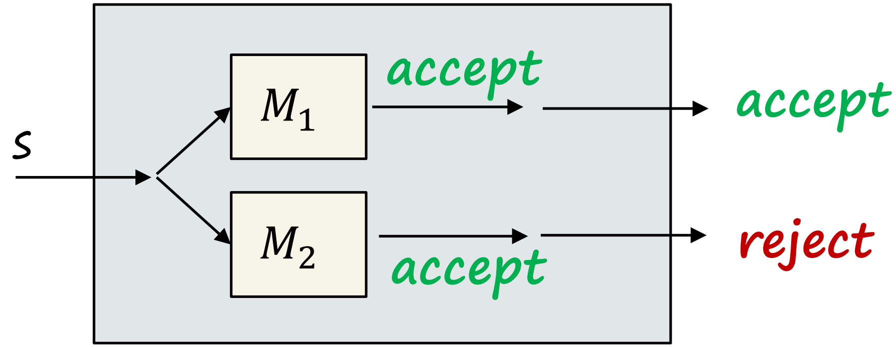


Complements of Language

If both $\boldsymbol{L}$ and $\overline{\boldsymbol{L}}$ is RE, then $\boldsymbol{L}$ is recursive

Proof:

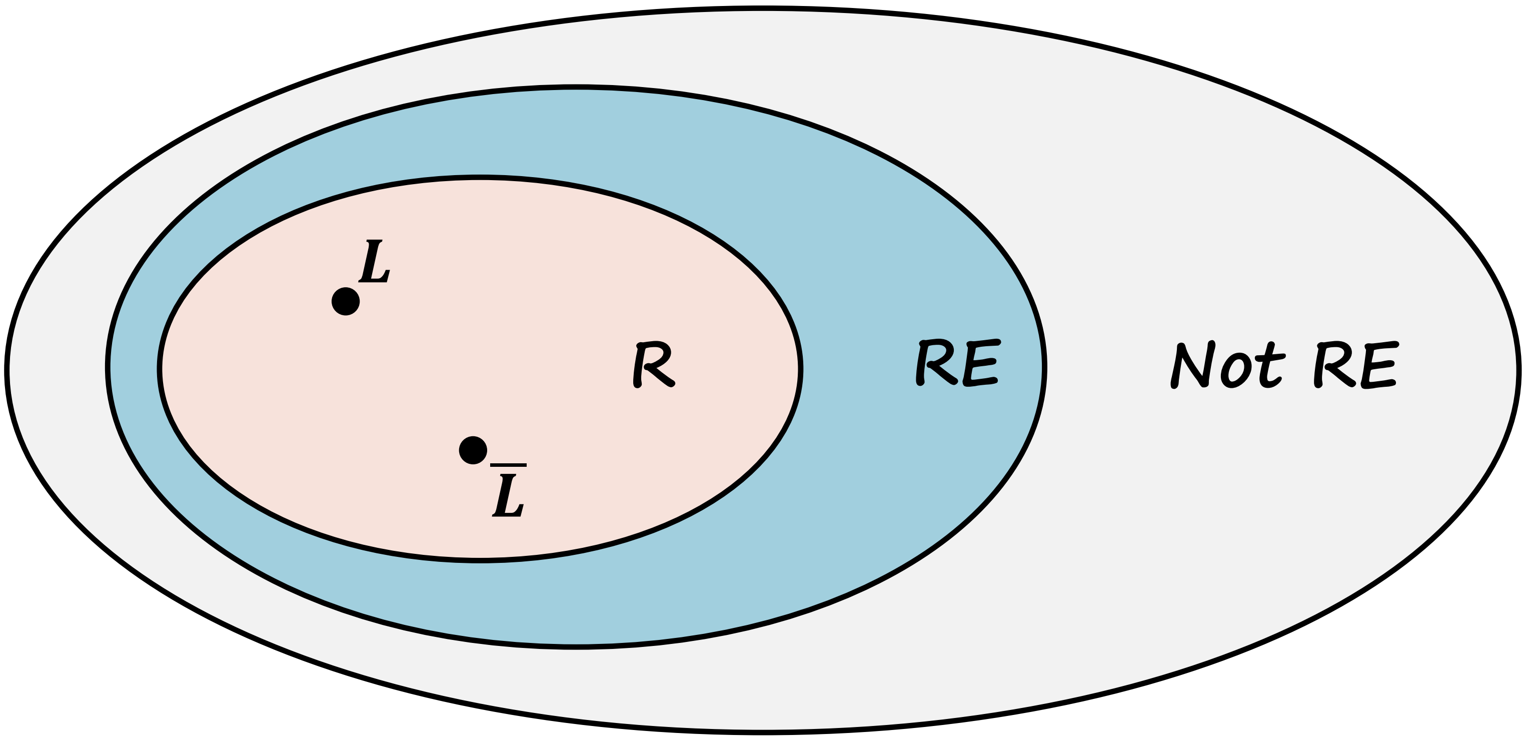
- Let $L = L(M_1)$ and $\bar{L} = L(M_2)$ for some TM M_1 and M_2. We construct a TM M_3 that always halts such that $L = L(M_3)$ by combining M_1 and M_2



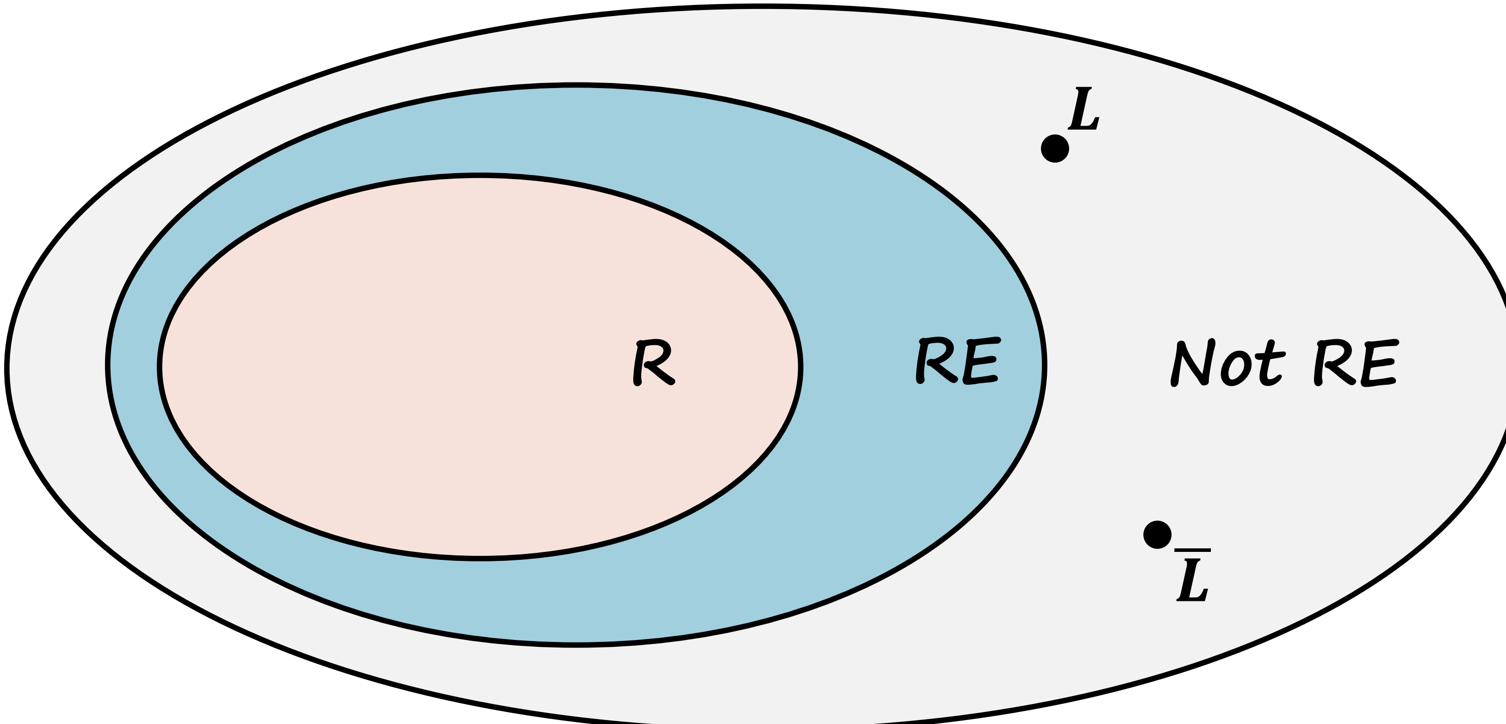
Complements of Language

- For any language L, L can be one of three kinds types: **R**, **RE** but not **R** (If any), not **RE.**
- But there are only four possible types combination of L and $\overline{L}$
  - Both L and $\overline{L}$ are R
  - Both L and $\overline{L}$ are not RE
  - One of L and $\overline{L}$ is RE, the other one is not RE

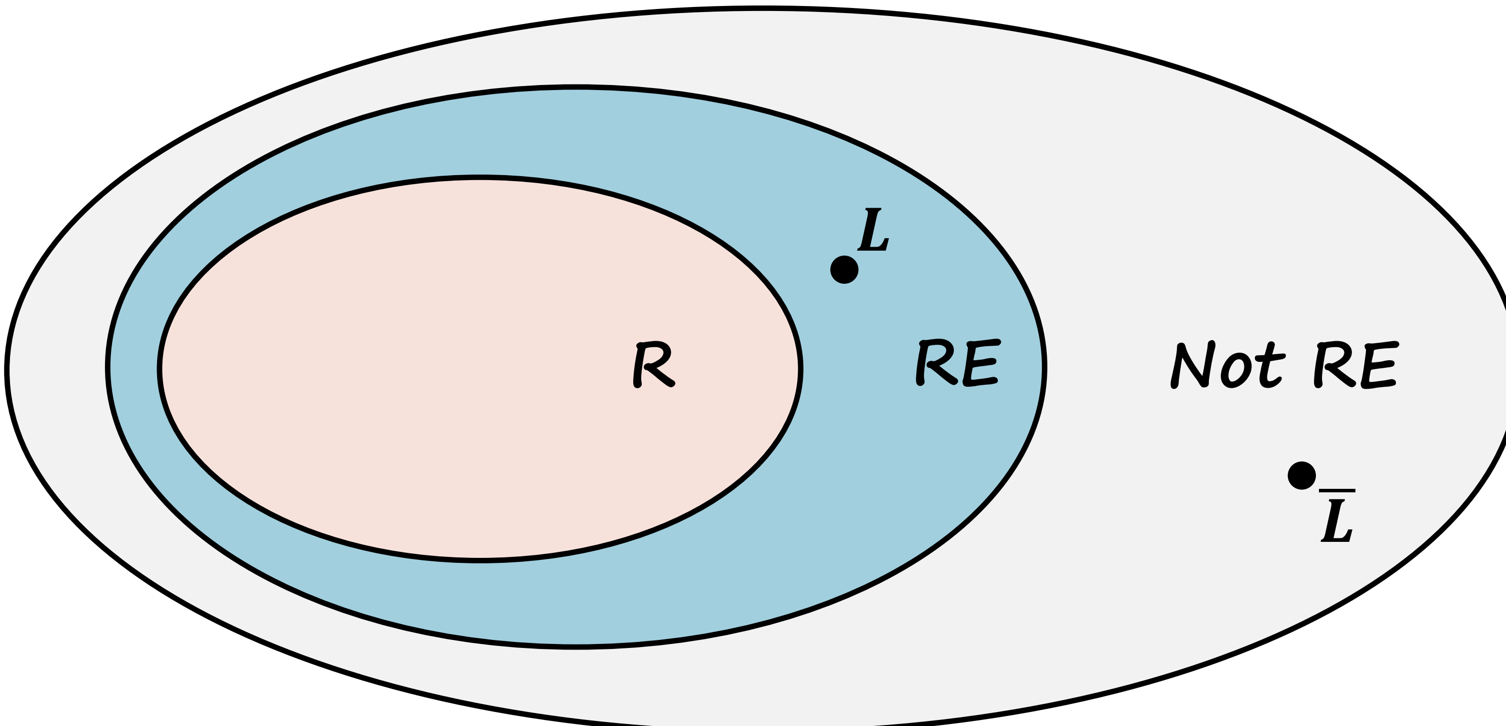
Both in Recursive



Both in Not RE



Both in Not RE

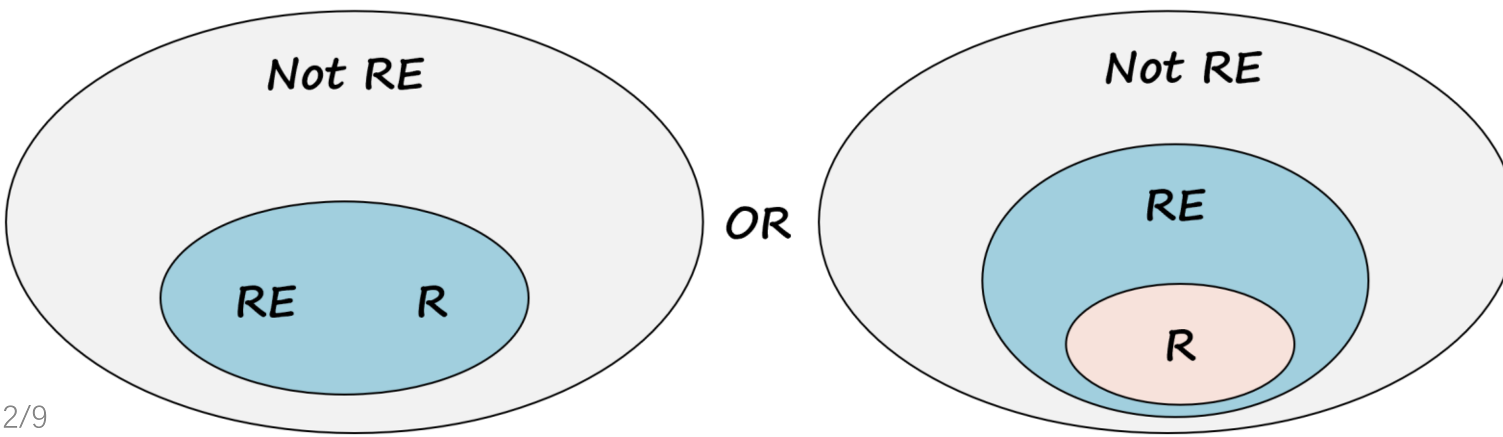


R & RE

- We've know $\mathbf{R} \subseteq \mathbf{RE}$, but more important question is that:

$$\mathbf{R} = \mathbf{RE}?$$

- That is, if you can just confirm "yes" answers to a problem, can you necessarily solve that problem?



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 $0^n 1^n$
- 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

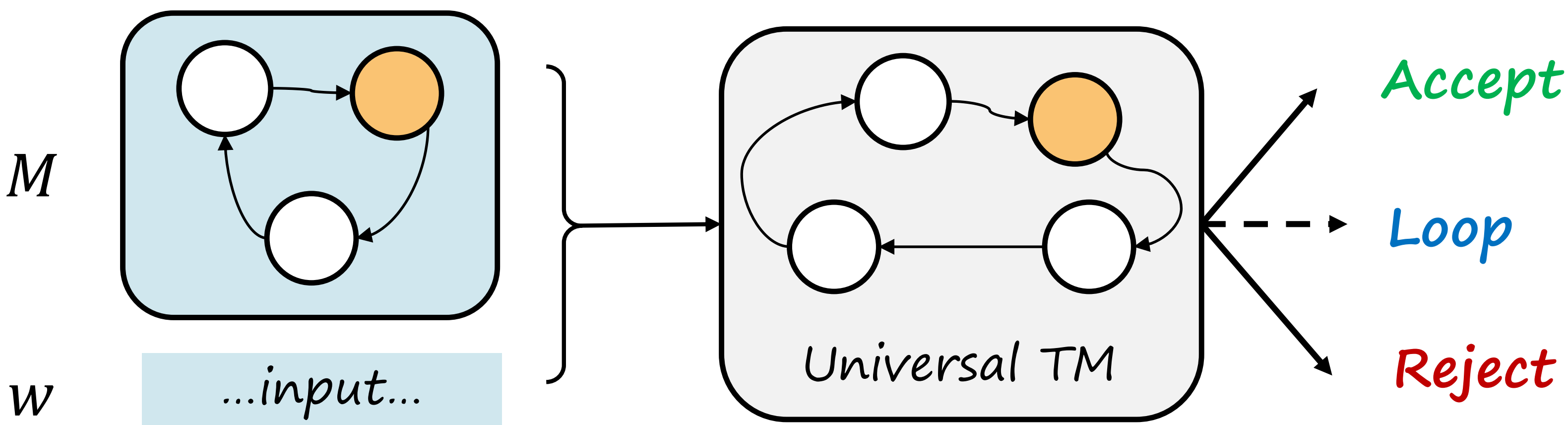
- We've known it is possible to program a TM simulator on an unbounded-memory computer.
- We could imagine it as a method

$$\text{boolean simulateTM(TM } M, \text{ string } w)$$

- 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

- It is also known that 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

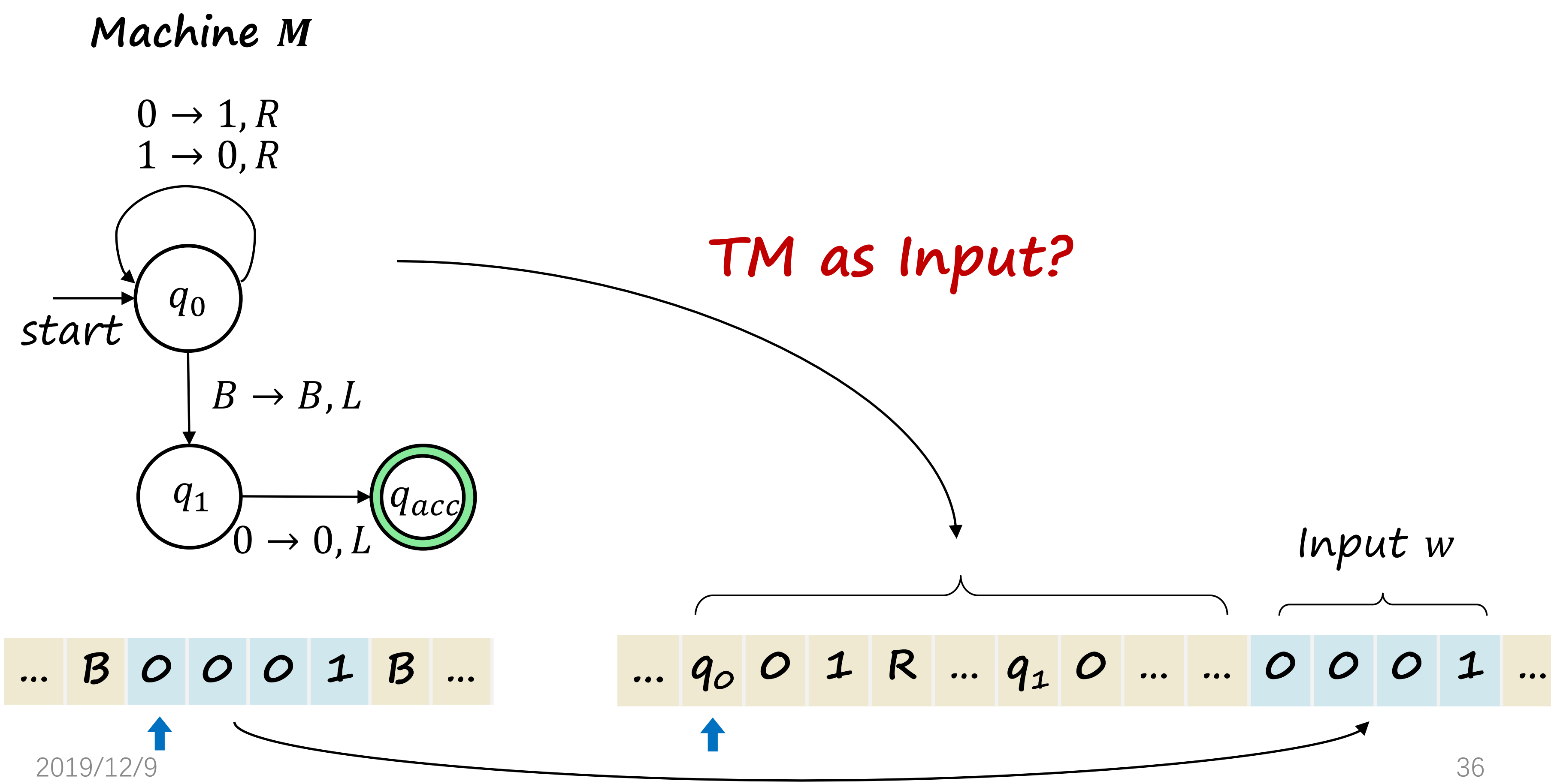
- **Theorem (Turing, 1936):** There is a Turing machine $\boldsymbol{U_{TM}}$ called **the Universal Turing Machine** that, when run on an input of the form $\langle M, w \rangle$, where M is a Turing machine and w is a string, simulates M running on w and **does whatever** M does on w (accepts, rejects, 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 $\boldsymbol{\delta}(\boldsymbol{q}_{\boldsymbol{i}}, \boldsymbol{X}_{\boldsymbol{j}}) = (\boldsymbol{q}_{\boldsymbol{k}}, \boldsymbol{X}_{\boldsymbol{l}}, \boldsymbol{D}_{\boldsymbol{m}})$, we shall code this rule by the string $\mathbf{0}^{\boldsymbol{i}}\mathbf{10}^{\boldsymbol{j}}\mathbf{10}^{\boldsymbol{k}}\mathbf{10}^{\boldsymbol{l}}\mathbf{10}^{\boldsymbol{m}}$.
  - Notice $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}, \boldsymbol{l}, \boldsymbol{m}$ 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 $\boldsymbol{C}_{\boldsymbol{i}}$ 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

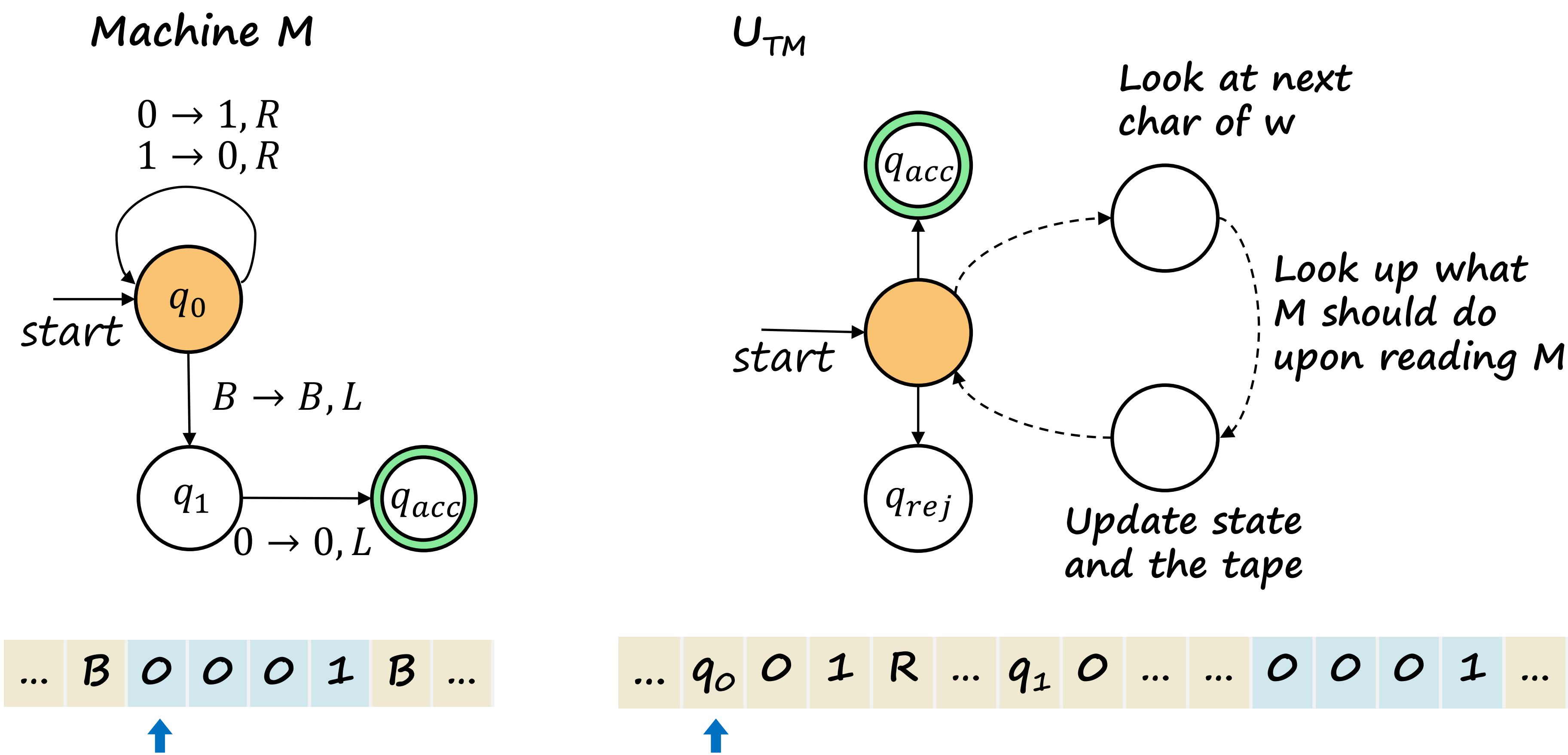
- Let a TM: $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1, B\}, \{\delta\}, q_1, B, q_2)$
  - $X_1 = 0, X_2 = 1, X_3 = B, D_1 = L, D_2 = R$
- Transition Function δ**:**
  - $\boldsymbol{\delta(q_1, 1) = (q_3, 0, R)}$
  - $\boldsymbol{\delta(q_3, 0) = (q_1, 1, R)}$
  - $\boldsymbol{\delta(q_3, 1) = (q_2, 0,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

Example of Code for TM

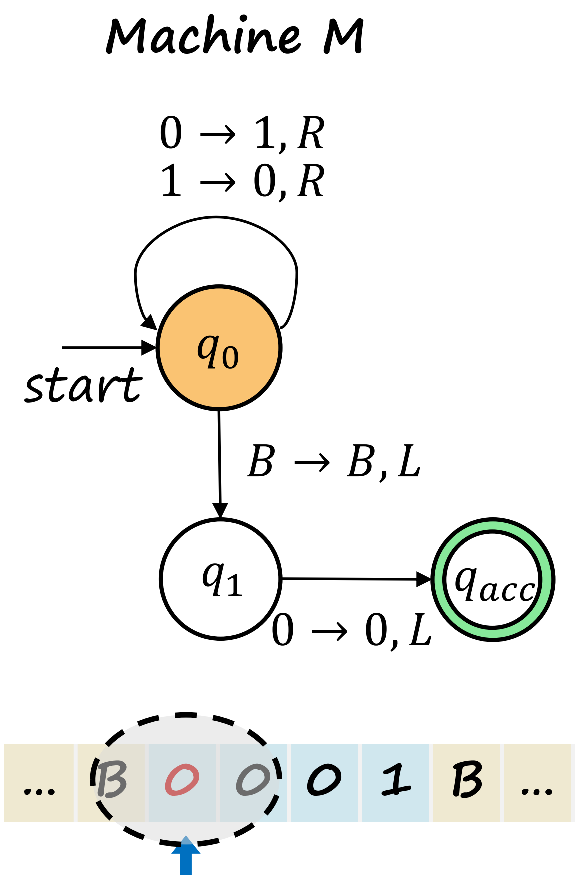
- Let a TM: $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1, B\}, \{\delta\}, q_1, B, q_2)$
  - $X_1 = 0, X_2 = 1, X_3 = B, D_1 = L, D_2 = R$
- Transition Function δ:
  - $\boldsymbol{\delta(q_1, 1) = (q_3, 0, R) \Rightarrow \delta(q_1, X_2) = (q_3, X_1, D_2) \Rightarrow 0100100010100}$
  - $\boldsymbol{\delta(q_3, 0) = (q_1, 1, R) \Rightarrow \delta(q_3, X_1) = (q_1, X_2, D_2) \Rightarrow 0001010100100}$
  - $\boldsymbol{\delta(q_3, 1) = (q_2, 0, R) \Rightarrow \delta(q_3, X_2) = (q_2, X_1, D_2) \Rightarrow 00010010010100}$
  - $\boldsymbol{\delta(q_3, B) = (q_3, 1, L) \Rightarrow \delta(q_3, X_3) = (q_3, X_2, D_1) \Rightarrow 0001000100010010}$
- **Code for this TM:**

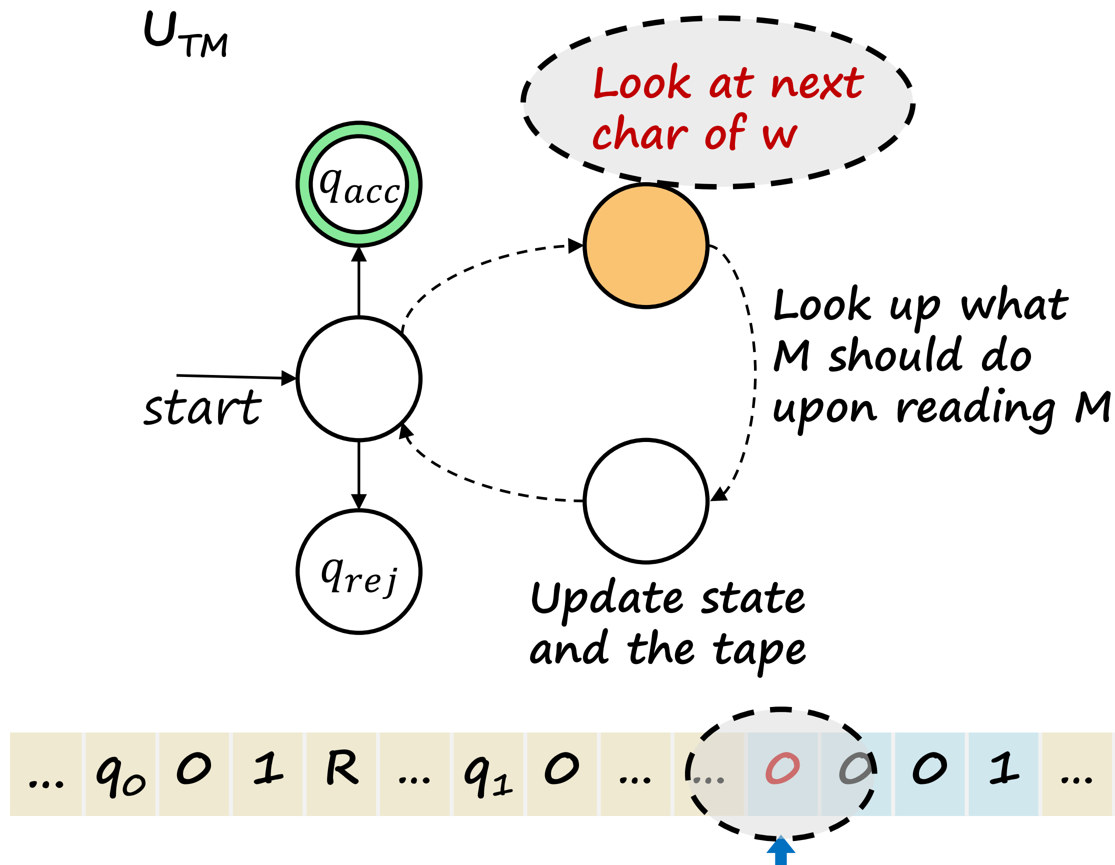
$$\mathbf{0100100010100110001010100100110001001001010011 0001000100010010}$$

The Universal Turing Machine

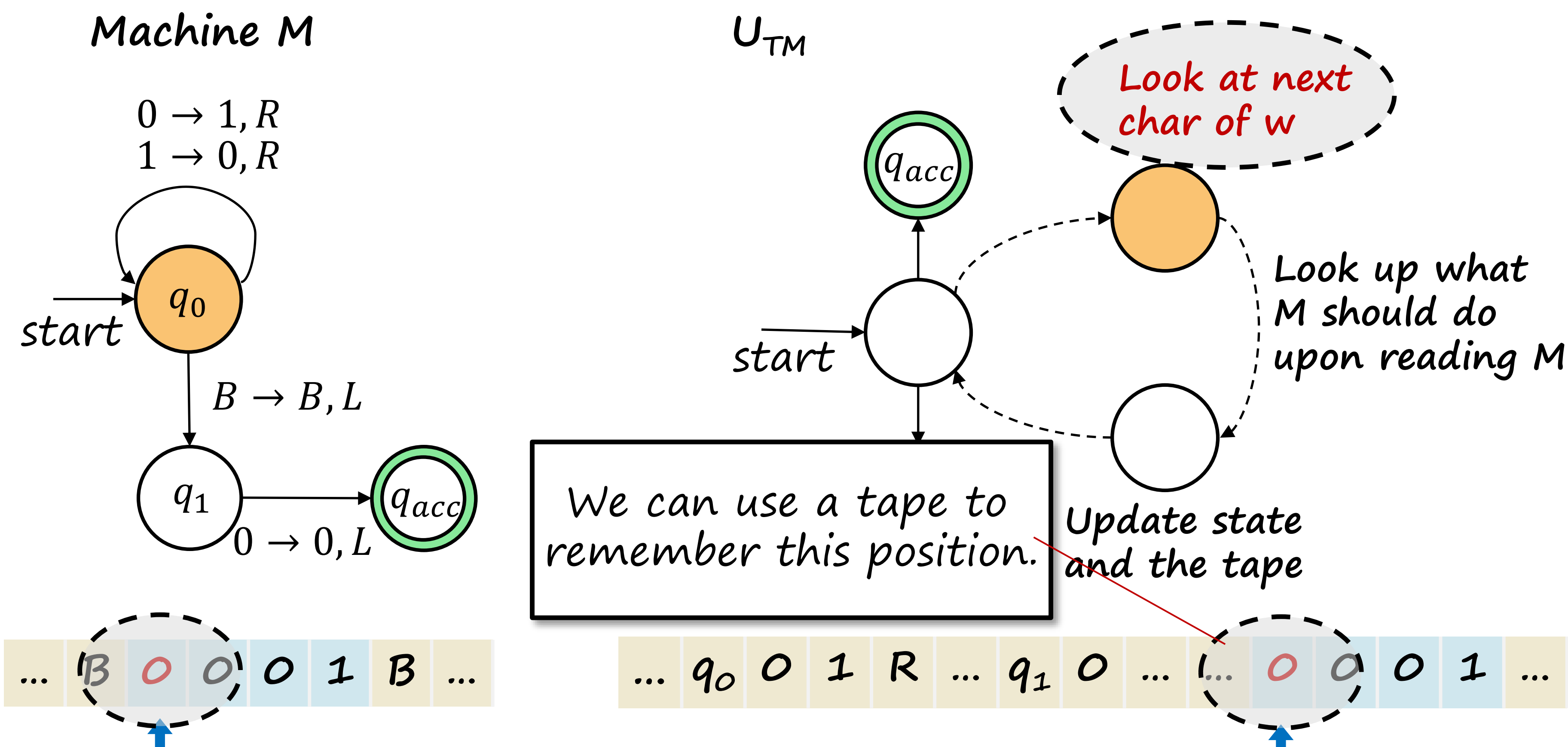


The Universal Turing Machine

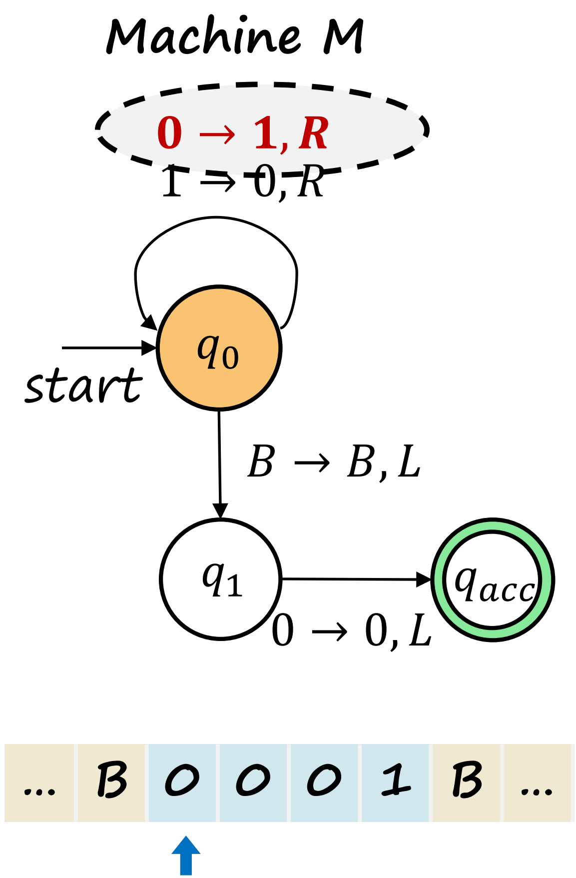


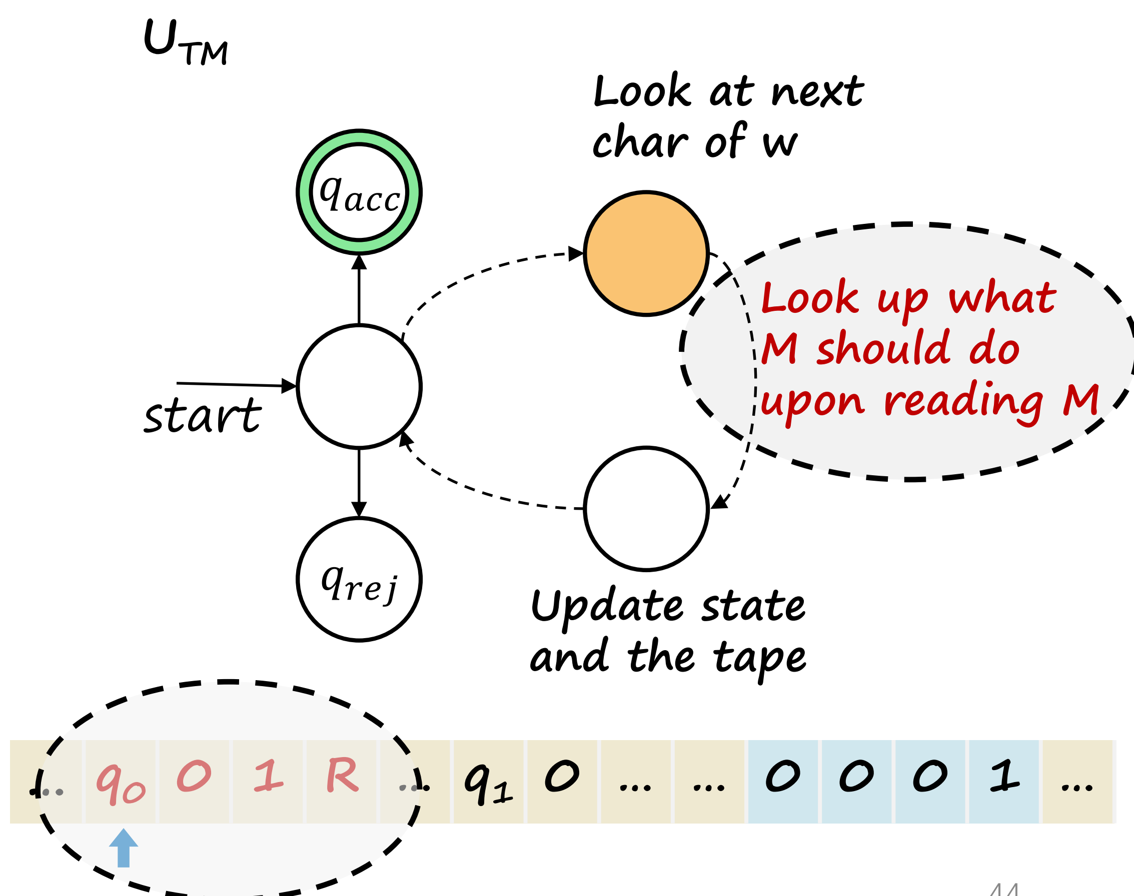


The Universal Turing Machine

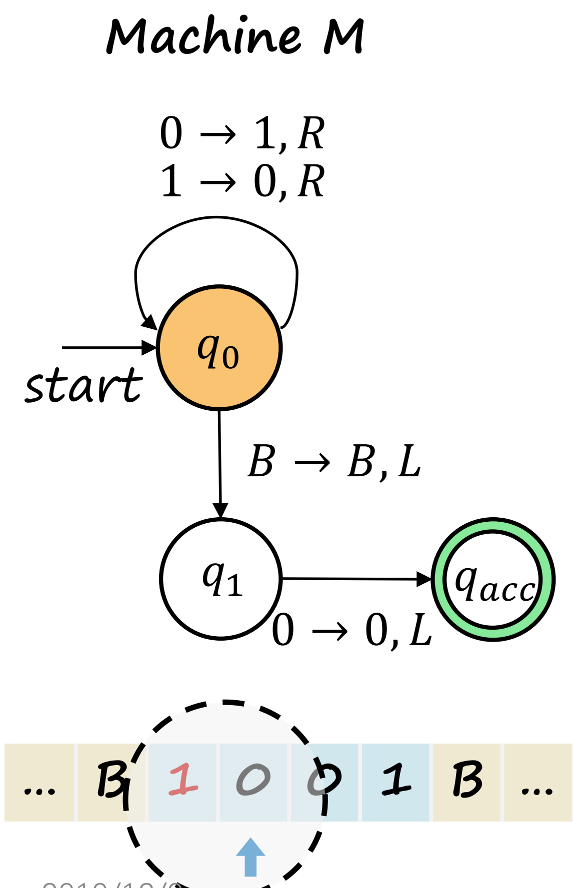


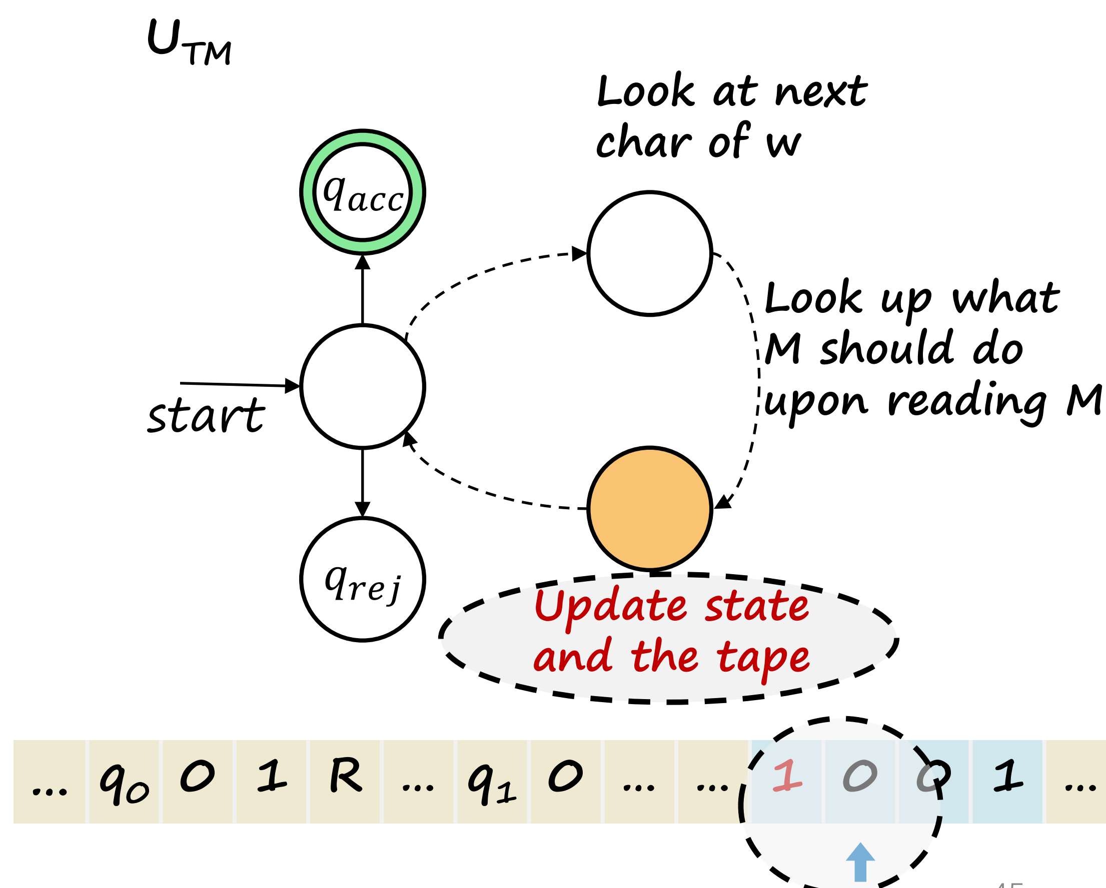
The Universal Turing Machine



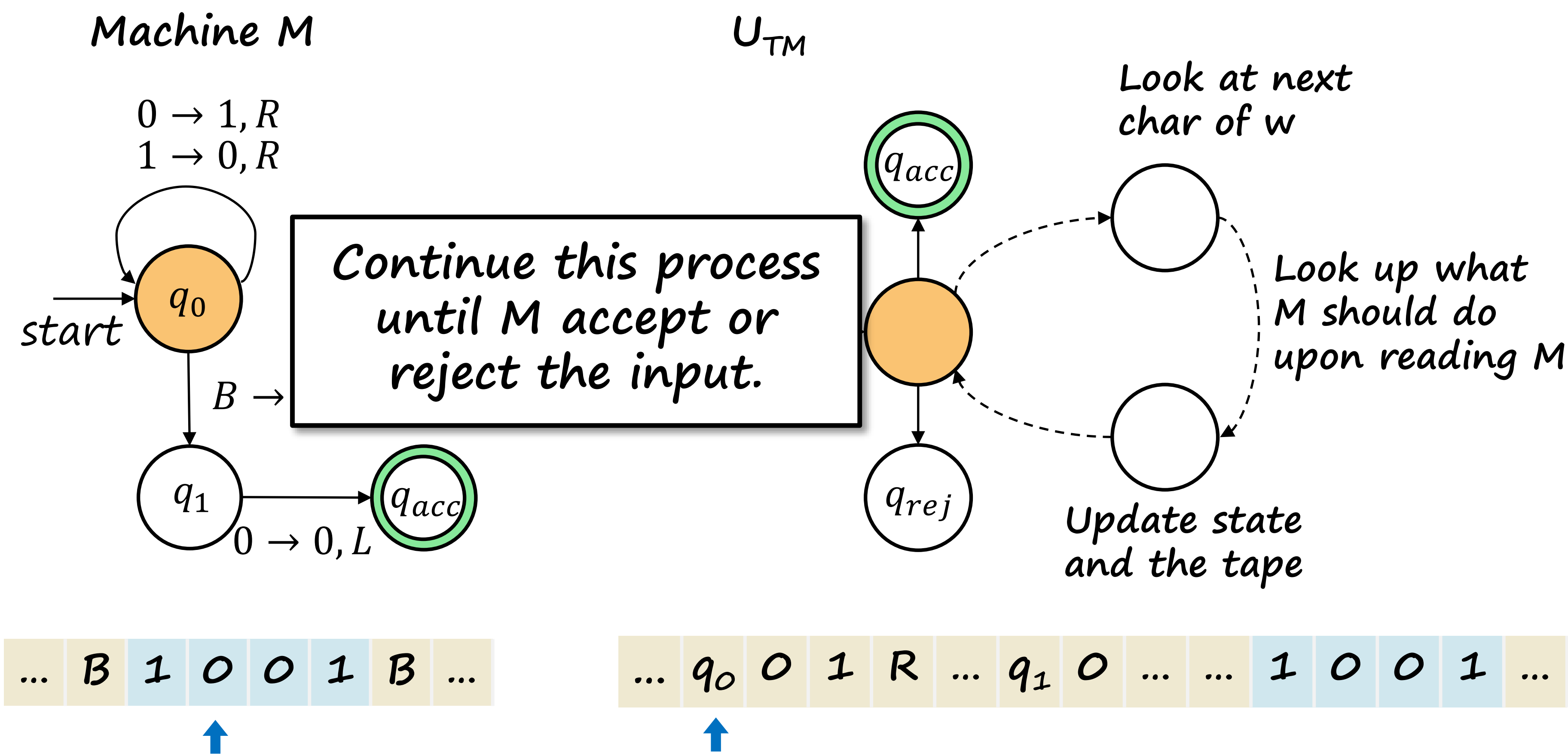


The Universal Turing Machine

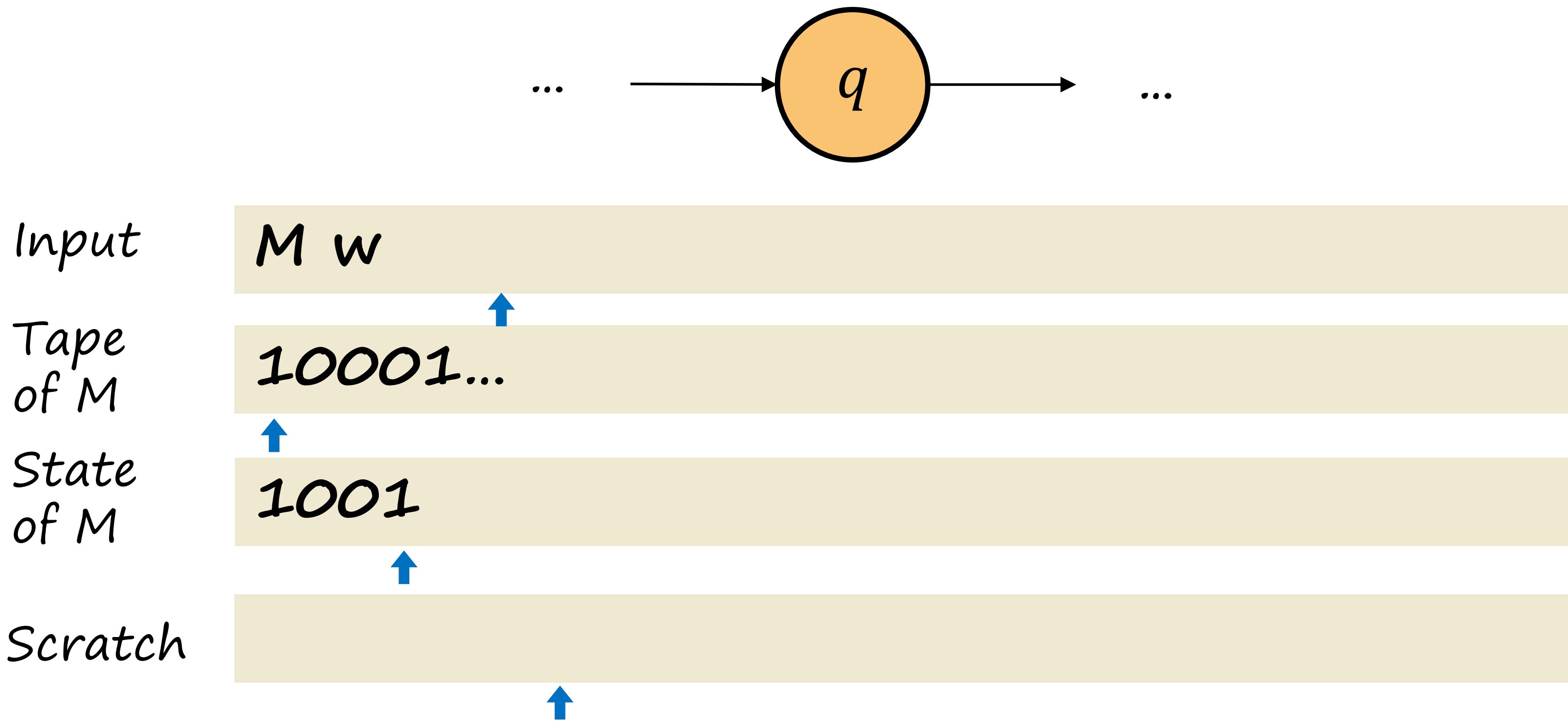




The Universal Turing Machine



The Universal Turing Machine



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}

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

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

- Useful fact:

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

- 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 $\left\langle U_{TM}, \langle M, w\rangle\right\rangle$
  - U_{TM} accepts $\left\langle U_{TM}, \left\langle U_{TM}, \langle M, w\rangle\right\rangle\right\rangle$
  - ….

Next

Self-Reference

- Turing machines that compute on themselves!

Undecidable Problems

- Problems truly beyond the limits of algorithmic problem-solving!


THERE'S STILL
HOPE

测试范围

两部分：

- 形式化验证 （70%）
- 图灵机（30%）

测试范围

形式化验证部分：70%

- 命题逻辑：程序转化，DPLL算法
- 谓词逻辑：程序转化，lazy SMT techniques，EUF solver，Nelson-Oppen method，trigger matching
- 霍尔逻辑：公理和推导规则，循环不变量，最弱前置条件

不考的部分：

Switch variable，GSAT，eager SMT techniques，对lazy SMT techniques的优化(incremental T-solver，theory propagation)，e-matching，公理系统，最强后置条件

测试范围

图灵机部分：30%

- 图灵机基础：DFA；图灵机定义、表示、计算；设计图灵机解决某一问题。RE&R的定义，性质（交并补）
- 不可判定性：不可判定问题（通用图灵机语言、停机问题）；了解构造证明和规约。

期末考试范围

课程组统一出卷阅卷

- 命题范围: 逻辑1,2,4,5章
- 集合论9,10,11章