Chapter 7: Turing Machines

Dr. Yuan Yao

University of Nottingham Ningbo China (UNNC)

Learning Outcomes

Learning outcomes

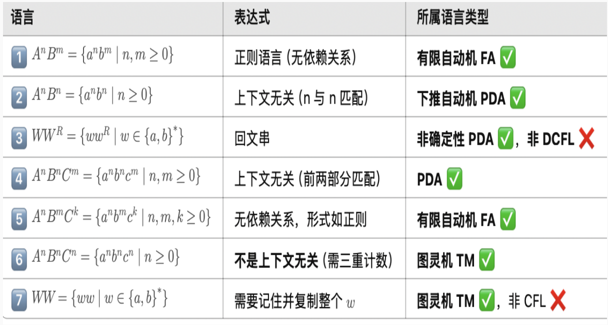
At the conclusion of this chapter, the students are expected to be able to:

- Describe the components of a standard Turing machine.
- State whether an input string is accepted by a Turing machine.
- Construct a Turing machine to accept a specific language.
- Trace the operation of a Turing machine transducer given a sample input string.
- Construct a Turing machine to compute a simple function.
- State Turing's thesis and discuss the circumstantial evidence supporting it.
- Understand the Turing Machine halting problem and how to prove it.

Introduction

Models of Computation

- Given the following languages, which models of computations we can use to accept them?
    - $A^nB^m = \{a^nb^m \mid n, m \geq 0\}$
    - $A^nB^n = \{a^nb^n \mid n \geq 0\}$
    - $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$
    - $A^nB^nC^m = \{a^nb^nc^m \mid n, m \geq 0\}$
    - $A^nB^mC^k = \{a^nb^mc^k \mid n, m, k \geq 0\}$
    - $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$
    - $WW = \{ww \mid w \in \{a, b\}^*\}$



语言	表达式	所属语言类型
1 $A^nB^m = \{a^nb^m \mid n, m \geq 0\}$	正则语言 (无依赖关系)	**有限自动机 FA** ✅
2 $A^nB^n = \{a^nb^n \mid n \geq 0\}$	上下文无关 (n 与 n 匹配)	**下推自动机 PDA** ✅
3 $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$	回文串	**非确定性 PDA** ✅, **非 DCFL** ❌
4 $A^nB^nC^m = \{a^nb^nc^m \mid n, m \geq 0\}$	上下文无关 (前两部分匹配)	**PDA** ✅
5 $A^nB^mC^k = \{a^nb^mc^k \mid n, m, k \geq 0\}$	无依赖关系，形式如正则	**有限自动机 FA** ✅
6 $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$	**不是上下文无关 (需三重计数)**	**图灵机 TM** ✅
7 $WW = \{ww \mid w \in \{a, b\}^*\}$	需要记住并复制整个 w	**图灵机 TM** ✅, 非 CFL ❌

A General Model of Computation

- Both finite automata and pushdown automata are models of computation.
- However, there are languages which are not accepted by these models:
  - An FA cannot accept languages such as:
    - $A^nB^n = \{a^nb^n \mid n \geq 0\}$ 需要计数匹配（超出 FA 能力）
    - $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$ 回文需要记忆整个前缀（FA 没有栈）
  - An PDA cannot accept languages such as:
    - $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$ 需要同时匹配三种符号数量，PDA 栈结构无法实现两级嵌套依赖
    - $WW = \{ww \mid w \in \{a, b\}^*\}$ 字符串重复，需要存储并完整对比原始串，超出 PDA 的“ 上下文无关 ”能力
- However, these tasks can be performed by programs written in general purpose programming languages such as Java, C, or Haskell. 现实中的编程语言具备图灵完备性，能力等价于图灵机。
- Therefore, FAs and PDAs do not provide us with the most powerful computational power. **FA 和 PDA 都不是最强的计算模型。**
- **Question: Which model of computation (if any) can provide us with the most powerful computational power?**

A General Model of Computation

- In the first part of the 20th century, the most prominent mathematicians and logicians were occupied by the previous question, which is closely related to the following:
  - What are the limits of algorithmic thinking? **算法性思维的极限是什么？**
- To address this question, Alan Turing considered a human computer working with a pen and paper.
  - In those days, the word “computer” referred to people (i.e., human beings) who would work in an office, and perform long computations, using special purpose (and at times, ingenious) computing devices.

人类计算员坐在办公室，用纸笔一步步执行规则，比如计算乘法表、做银行账簿等。

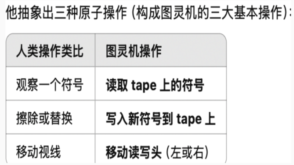
图灵机的本质是对 “ 人类执行算法 ” 的最基本形式化抽象。即：**我们如何把人类的计算行为变成可以机械执行的步骤。**

A General Model of Computation

- Turing considered the very basic operations performed when a human computer solves a problem algorithmically, by using pen and paper.

图灵观察的是人类在纸上算法性解题的**最基本动作**

- He postulated that the steps a (human) computer takes should include these:
  - Examine an individual symbol on the paper.
  - Erase a symbol or replace it by another.
  - Transfer attention from one symbol to a nearby one.


他抽象出三种原子操作（构成图灵机的三大基本操作）:

人类操作类比	图灵机操作
观察一个符号	**读取 tape 上的符号**
擦除或替换	**写入新符号到 tape 上**
移动视线	**移动读写头**（左或右）

- Based on that, he came up with his ground-breaking idea regarding how a universal computing machine should work.
- Turing's objective was to demonstrate the inherent **limitations** of algorithmic methods.
- This is why he wanted his device to be able to execute **any** algorithm that a human computer could.

A General Model of Computation

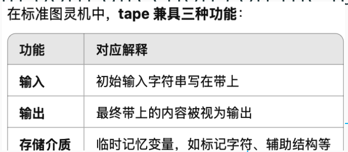
- In simple terms, a Turing machine is a finite state control unit with an unbounded tape attached as a storage device. 图灵机 = **状态机 + 无限内存**
- Even though a Turing machine is a very simple machine, it turns out to be very powerful, and has been the basis of languages such as C, Python, Java...
- In fact, so far, no one has been able to find any other effective model of computation more powerful than the Turing machine, although many models have proven to be as powerful as that of Turing machines. **目前没有更强的模型**

  要么等价于图灵机（同样强）要么更弱（无法实现所有可计算函数）
- This has led to Church-Turing Thesis, which claims that Turing machines are the most general types of automata, in principle as powerful as any computers. 任意**人类或机器能够完成的算法性任务**，都可以被某台图灵机完成。
  - We will discuss it later.

Turing Machine

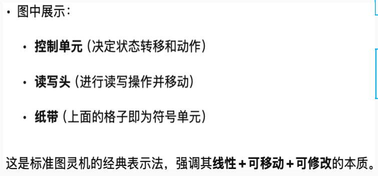
The Standard Turing Machine

- A standard Turing machine has unlimited storage in the form of a tape:
  - The tape consists of an infinite number of cells.
  - With each cell capable of storing one symbol.
- The *read-write head* can travel in both directions on the tape, processing one symbol per move.

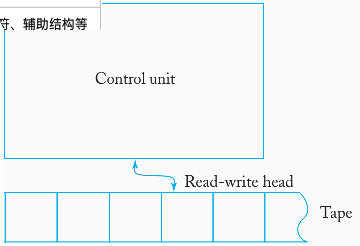

在标准图灵机中，**tape 兼具三种功能**：

功能	对应解释
输入	初始输入字符串写在带上
输出	最终带上的内容被视为输出
存储介质	临时记忆变量，如标记字符、辅助结构等

- In a Standard Turing Machine, the tape acts as the:
  - input
  - output
  - and storage medium

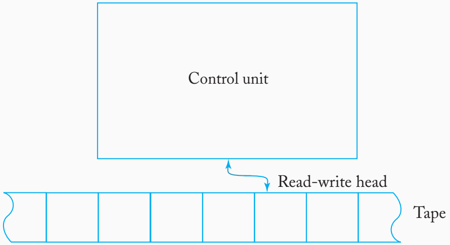

- 图中展示：
  - **控制单元**（决定状态转移和动作）
  - **读写头**（进行读写操作并移动）
  - **纸带**（上面的格子即为符号单元）

这是标准图灵机的经典表示法，强调其**线性＋可移动＋可修改**的本质。



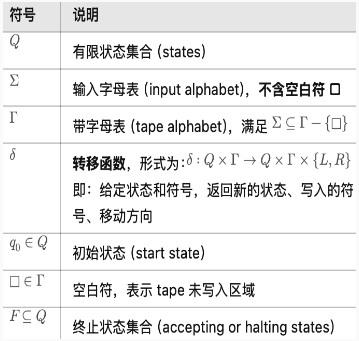
The Standard Turing Machine

- A control function causes the machine to change states and possibly overwrite the tape contents. 切换状态（change state）；修改带上的符号（overwrite tape contents）
- Deterministic and non-deterministic Turing machines turn out to be as powerful as each other. **确定性与非确定性图灵机是等价的**
- The input string is surrounded by blanks, so the input alphabet is considered a proper subset of the tape alphabet.



Definition of a Turing Machine

- **Definition 9.1:** A Turing Machine $M = \{Q, \Sigma, \Gamma, \delta, q_0, \Box, F\}$ is defined by:
  - Q : a finite set of internal states.
  - Σ : the input alphabet.
  - Γ : the tap alphabet.
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: the transition function.
  - $\Box \in \Gamma$: a special symbol called the blank.
  - $q_0 \in Q$: the initial state.
  - $F \subseteq$ the set of final states.



符号	说明
Q	有限状态集合 (states)
Σ	输入字母表 (input alphabet)，**不含空白符 □**
Γ	带字母表 (tape alphabet)，满足 $\Sigma \subseteq \Gamma - \{\Box\}$
δ	**转移函数**，形式为：$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ 即：给定状态和符号，返回新的状态、写入的符号、移动方向
$q_0 \in Q$	初始状态 (start state)
$\Box \in \Gamma$	空白符，表示 tape 未写入区域
$F \subseteq Q$	终止状态集合 (accepting or halting states)

- In the definition of a Turing machine, we assume that:

$$\Sigma \subseteq \Gamma - \{\Box\}$$

Transition function δ

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

给定**当前状态与当前读到的带符号**，图灵机会：
转换到一个新的状态
替换 tape 上的符号
决定读写头移动方向（左 L 或 右 R）

- Input to the transition function δ consists of:
  - the current state of the control unit.
  - and the current tape symbol
- Output of δ consists of :
  - a new state
  - a new tap symbol
  - and location of the next symbol to be read (left or right)
- In general, δ is a partial function, so on some (state, symbol) input combinations it may be undefined.

δ 是一个偏函数（partial function）
不是所有组合的输入都有定义
若某个状态-符号组合没有定义，图灵机在此配置将**停止（halt）**

Example: δ transition

- The following is a sample transition rule in a Turing machine:

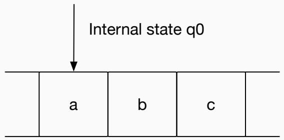
$$\delta(q_0, a) = (q_1, d, R)$$

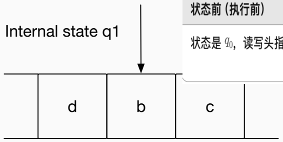
- According to the rule, when:
  - the control unit is in state q_0,
  - and the tape symbol is a

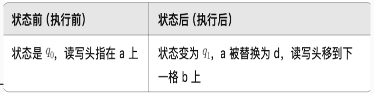
当前状态是 q_0
当前读到的 tape 符号是 a

- then, the effect of the transition is as follows:
  - the new state will be q_1,
  - the symbol d replaces a on the tape,
  - and the read-write head moves one cell to the right.

切换到**新状态** q_1；
将 tape 上的 a 替换为 d；
向右移动一格（R = Right）。







状态前（执行前）	状态后（执行后）
状态是 q_0，读写头指在 a 上	状态变为 q_1，a 被替换为 d，读写头移到下一格 b 上

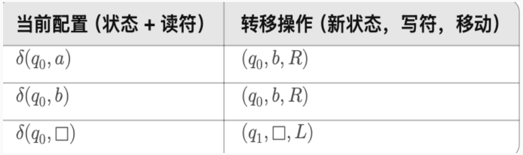
Example: Turing Machine

- Consider the Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ with:

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{a, b, \Box\}, F = \{q_1\}$$

- The transition function δ is given by:

$$\begin{aligned} \delta(q_0, a) &= (q_0, b, R) \\ \delta(q_0, b) &= (q_0, b, R) \\ \delta(q_0, \Box) &= (q_1, \Box, L) \end{aligned}$$

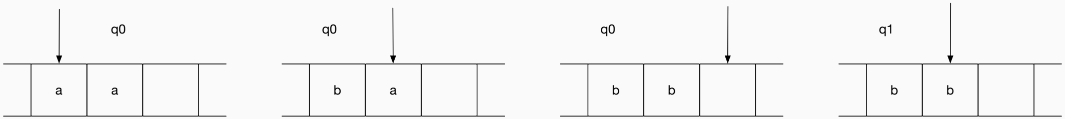


当前配置（状态 + 读符）	转移操作（新状态，写符，移动）
$\delta(q_0, a)$	(q_0, b, R)
$\delta(q_0, b)$	(q_0, b, R)
$\delta(q_0, \Box)$	$(q_1, \Box, L)$

- Can you find out what this TM does?

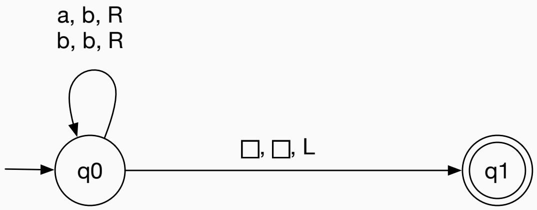
Example: Turing Machine

- The machine starts in q_0 and, as long as it reads $a's$, it will replace them with $b's$ and continue moving to the right, but $b's$ will not be modified.
- When a blank is found, the control unit switches states to the final state q_1 and moves one cell to the left. **将所有的 a 替换为 b，不动原有的 b，读到空白就停机。**
- The followings show a sequence of moves for the given Turing machine with the initial content aa:



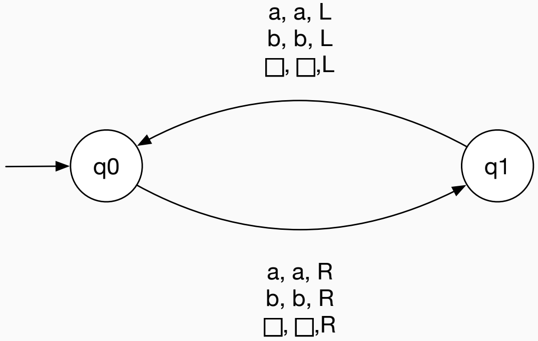
Transition Graphs for Turing Machines

- As before, we can use transition graphs to represent Turing machines.
- We label the edges of the graph with three items:
    - the current tape symbol,
    - the symbol that replaces it,
    - and the direction in which the read-write head is to move.
- The Turing machine in the previous slides can be represented as follows:

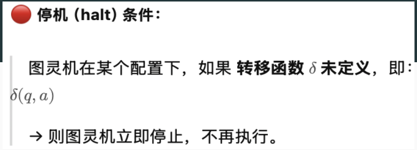


Another Example of Turing Machines

- Given the following transition graph of a Turing Machine, what happens if the input string is ab (starting from the initial state q_0)?



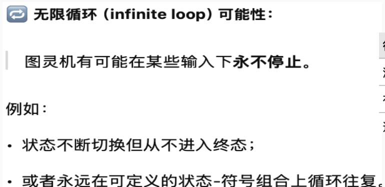
Another Example of Turing Machines



🔴 **停机（halt）条件：**

> 图灵机在某个配置下，如果 **转移函数** δ **未定义**，即：$\delta(q, a)$
>
> → 则图灵机立即停止，不再执行。

- A Turing machine is said to halt whenever it reaches a configuration for which δ is not defined.
- It is possible for a Turing machine to never halt on certain inputs.
- It should be clear that this machine will run forever, regardless of the initial information on its tape, with the read-write head moving alternately right then left but making no modifications to the tape.
- In analogy with programming terminology, we say that the Turing machine is in an infinite loop.

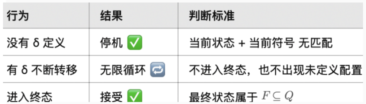


🔁 **无限循环（infinite loop）可能性：**

> 图灵机有可能在某些输入下**永不停止。**

例如：

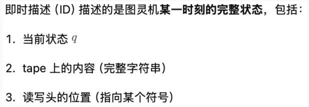
- 状态不断切换但从不进入终态；
- 或者永远在可定义的状态-符号组合上循环往复。



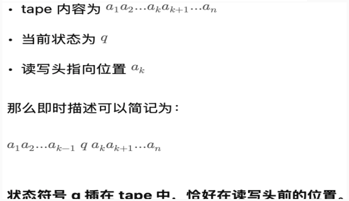
行为	结果	判断标准
没有 δ 定义	停机 ✅	当前状态 + 当前符号 无匹配
有 δ 不断转移	无限循环 🔁	不进入终态，也不出现未定义配置
进入终态	接受 ✅	最终状态属于 $F \subseteq Q$

Instantaneous Description

- Here, as in the case of PDAs, the most convenient way to exhibit a sequence of configurations of a Turing machine is based on the idea of an *instantaneous description*.
- Any configuration is completely determined by:
    - the current state of the control unit,
    - the contents of the tape,
    - and the position of the read-write head.


即时描述（ID）描述的是图灵机**某一时刻的完整状态**，包括：

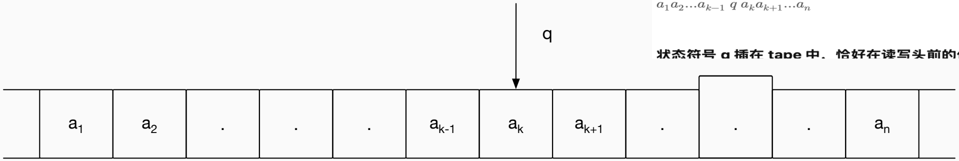
1. 当前状态 q
2. tape 上的内容（完整字符串）
3. 读写头的位置（指向某个符号）


- **tape 内容为** $a_1 a_2 \ldots a_k a_{k+1} \ldots a_n$
- **当前状态为** q
- **读写头指向位置** a_k

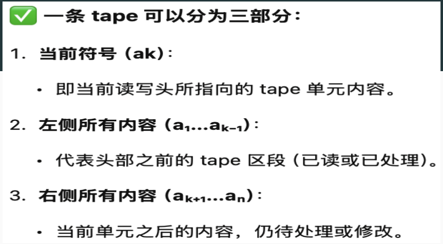
那么即时描述可以简记为：

$a_1 a_2 \ldots a_{k-1} \; q \; a_k a_{k+1} \ldots a_n$

状态符号 q 插在 tape 中，恰好在读写头前的位置。

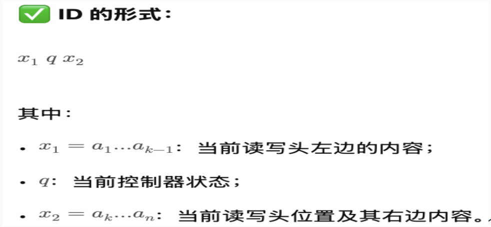


Instantaneous Description


✅ 一条 tape 可以分为三部分：

1. 当前符号（ak）：
   - 即当前读写头所指向的 tape 单元内容。
2. 左侧所有内容（$a_1 \ldots a_{k-1}$）：
   - 代表头部之前的 tape 区段（已读或已处理）。
3. 右侧所有内容（$a_{k+1} \ldots a_n$）：
   - 当前单元之后的内容，仍待处理或修改。

- A tape can be divided into three parts:
  - The current symbol (or cell), e.g., a_k.
  - All cells to the left, e.g., $a_1 a_2 \ldots a_{k-1}$.
  - All cells to the right, e.g., $a_{k+1} \ldots a_n$.
- An instantaneous description is in the form of $x_1 q x_2$, where
  - $x_1 = a_1 \ldots a_{k-1}$
  - q is the current state
  - $x_2 = a_k \ldots a_n$


✅ ID 的形式：

$x_1 \; q \; x_2$

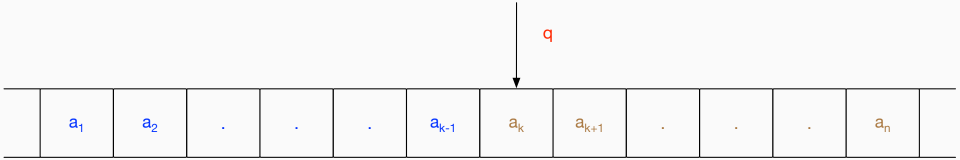
其中：

- $x_1 = a_1 \ldots a_{k-1}$：当前读写头左边的内容；
- q：当前控制器状态；
- $x_2 = a_k \ldots a_n$：当前读写头位置及其右边内容。

Instantaneous Description

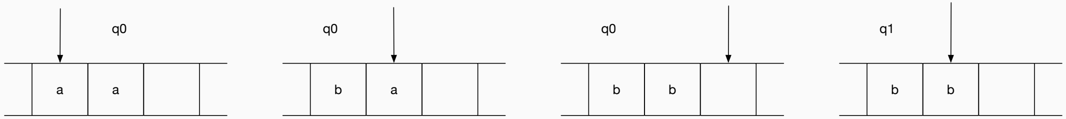
- An instantaneous description of a machine in state q with the tape depicted in the figure below is as follows:

$$a_1 a_2 \ldots a_{k-1} q a_k a_{k+1} \ldots a_n$$



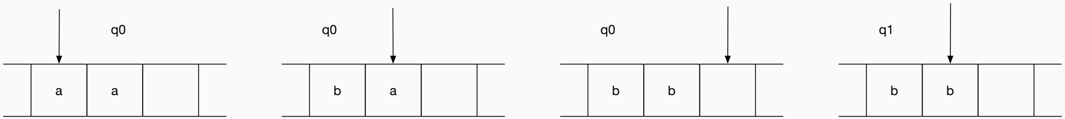
Instantaneous Description

- Write down the instantaneous description for the following Turing machine state.



Instantaneous Description

- Write down the instantaneous description for the following Turing machine state.
    - q_0aa
    - bq_0a
    - $bbq_0\square$
    - bq_1b



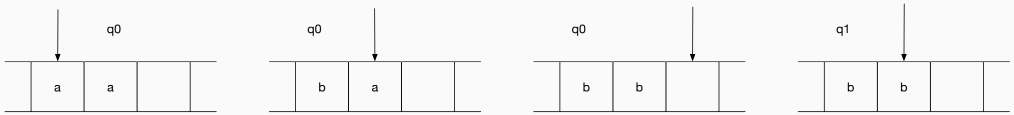
Instantaneous Description

- The sequence of moves in the following figure can therefore be represented as:

$$q_0aa \vdash bq_0a \vdash bbq_0\Box \vdash bq_1b$$

or

$$q_0aa \vdash^* bq_1b$$



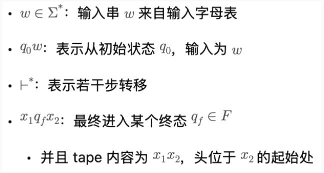
The Language Accepted by a Turing Machine

图灵机所接受的语言 L(M)，是所有**能够使图灵机停在终态**的输入串集合。

- Turing machines can be viewed as language accepters.
- **Definition** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a Turing Machine. Then the languages accepted by M is:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* x_1 q_f x_2\}$$

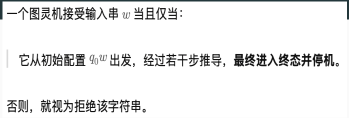
where $q_f \in F$ and $x_1, x_2 \in \Gamma^*$



- $w \in \Sigma^*$：输入串 w 来自输入字母表
- $q_0 w$：表示从初始状态 q_0，输入为 w
- $\vdash^*$：表示若干步转移
- $x_1 q_f x_2$：最终进入某个终态 $q_f \in F$
  - 并且 tape 内容为 $x_1 x_2$，头位于 x_2 的起始处

- The language accepted by a Turing machine is the set of all strings which cause the machine to halt in a final state, when started in its standard initial configuration.

The Language Accepted by a Turing Machine

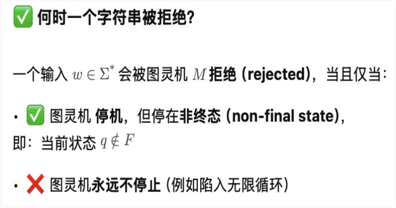


一个图灵机接受输入串 w 当且仅当：

> 它从初始配置 q_0w 出发，经过若干步推导，**最终进入终态并停机。**

否则，就视为拒绝该字符串。

- A string is rejected by M if:
  - The machine halts in a non-final state, or
  - The machine never halts.



✅ **何时一个字符串被拒绝？**

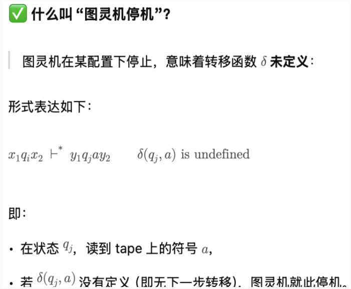
一个输入 $w \in \Sigma^*$ 会被图灵机 M **拒绝 (rejected)**，当且仅当：

- ✅ 图灵机 **停机**，但停在**非终态 (non-final state)**，
即：当前状态 $q \notin F$
- ❌ 图灵机**永远不停止**（例如陷入无限循环）

- M is also said to halt starting from some initial configuration $x_1q_ix_2$ if

$$x_1q_ix_2 \vdash^* y_1q_jay_2$$

for any q_j and a, for which $\delta(q_j, a)$ is undefined.



✅ **什么叫"图灵机停机"？**

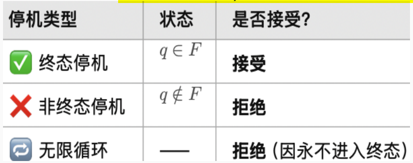
> 图灵机在某配置下停止，意味着转移函数 δ **未定义**：

形式表达如下：

$$x_1q_ix_2 \vdash^* y_1q_jay_2 \qquad \delta(q_j, a) \text{ is undefined}$$

即：

- 在状态 q_j，读到 tape 上的符号 a，
- 若 $\delta(q_j, a)$ 没有定义（即无下一步转移），图灵机就此停机。



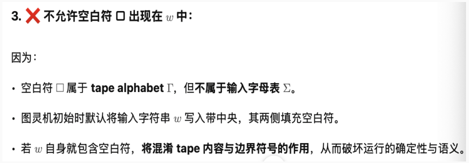
停机类型	状态	是否接受？
✅ 终态停机	$q \in F$	**接受**
❌ 非终态停机	$q \notin F$	**拒绝**
🔁 无限循环	——	**拒绝（因永不进入终态）**

The Language Accepted by a Turing Machine

- In the textbook, only the non-empty strings $w \in \Sigma^+$ are allowed. This has been done only for technical reasons to make some subsequent arguments easier and less tedious. We consider the more general case of $w \in \Sigma^*$, i.e., we allow the empty string as well.
- Note that, by allowing $w \in \Sigma^*$, we have excluded any string which included blank symbols.

- The previous definition indicates that the input w is written on the tape with blanks on either side.
- Why we exclude blanks from the input?

空白符是图灵机**控制带边界与未用空间的特殊标记**，为了防止语义歧义，**不能出现在输入串中**。



3. ❌ 不允许空白符 ☐ 出现在 w 中：

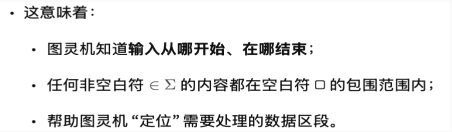
因为：

- 空白符 □ 属于 **tape alphabet** Γ，但**不属于输入字母表** Σ。
- 图灵机初始时默认将输入字符串 w 写入带中央，其两侧填充空白符。
- 若 w 自身就包含空白符，**将混淆 tape 内容与边界符号的作用**，从而破坏运行的确定性与语义。

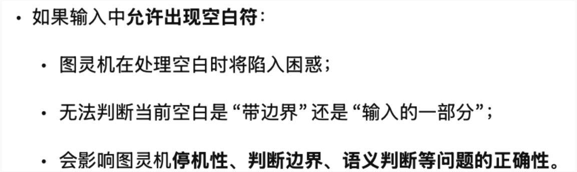
The Language Accepted by a Turing Machine

排除输入中出现空白符的理由，是为了保证图灵机能够明确识别输入的范围

- The reason for excluding blanks from the input now becomes clear:
  - It assures us that all the input is restricted to a well-defined region of the tape, bracketed by blanks on the right and left.
  - Without this convention, the machine could not limit the region in which it must look for the input; no matter how many blanks it saw, it could never be sure that there was not some nonblank input somewhere else on the tape.



- 这意味着：
  - 图灵机知道**输入从哪开始、在哪结束**；
  - 任何非空白符 $\in \Sigma$ 的内容都在空白符 $\square$ 的包围范围内；
  - 帮助图灵机“定位”需要处理的数据区段。



- 如果输入中**允许出现空白符**：
  - 图灵机在处理空白时将陷入困惑；
  - 无法判断当前空白是“带边界”还是“输入的一部分”；
  - 会影响图灵机**停机性、判断边界、语义判断等问题的正确性。**

Example: Languages Accepted by Turing Machines

- Consider the Turing Machine $M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, \Box\}, \delta, q_0, \Box, F)$ where δ is defined as follows:

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) \\ \delta(q_0, \Box) &= (q_1, \Box, R) \end{aligned}$$

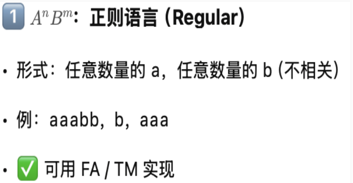
- What is the language $L(M)$?

$L(M) = \{0^n \mid n \geq 0\}$

即：所有只包含 0 的字符串，包括空串 ε

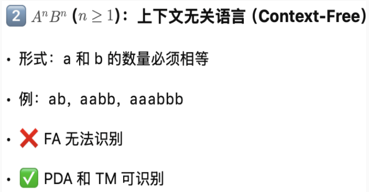
Example: Languages Accepted by Turing Machines

- Turing machines are more powerful than the previous classes of automata that we have studied (i.e., FAs and PDAs).
- Construct Turing machines for the following languages:
  - $A^n B^m$: which is regular.
  - $A^n B^n (n \geq 1)$: which is not regular but context-free.
  - $A^n B^n C^n (n \geq 1)$: which is not context-free.



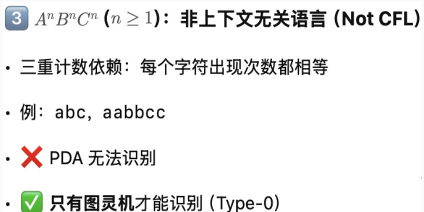
1️⃣ $A^n B^m$: **正则语言（Regular）**

- 形式：任意数量的 a，任意数量的 b（不相关）
- 例：aaabb，b，aaa
- ✅ 可用 FA / TM 实现



2️⃣ $A^n B^n$ $(n \geq 1)$：**上下文无关语言（Context-Free）**

- 形式：a 和 b 的数量必须相等
- 例：ab，aabb，aaabbb
- ❌ FA 无法识别
- ✅ PDA 和 TM 可识别



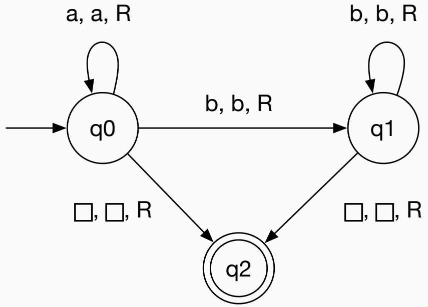
3️⃣ $A^n B^n C^n$ $(n \geq 1)$：**非上下文无关语言（Not CFL）**

- 三重计数依赖：每个字符出现次数都相等
- 例：abc，aabbcc
- ❌ PDA 无法识别
- ✅ **只有图灵机**才能识别（Type-0）

Example: Languages Accepted by Turing Machines

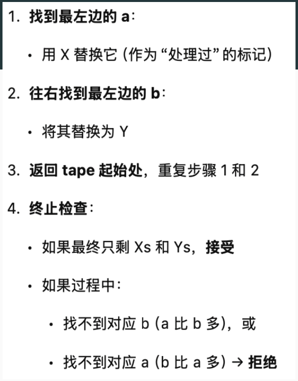
- Turing machines are more powerful than the previous classes of automata that we have studied (i.e., FAs and PDAs).

- Construct Turing machines for the following languages:
  - $A^n B^m$: which is regular.



Example: Languages Accepted by Turing Machines

- $A^n B^n (n \geq 1)$: which is not regular, but context-free.
  1. Find the leftmost *a*, replace it with some new symbol.
  2. Find the leftmost *b*, replace it with another symbol.
  3. Repeat step 1 and step 2.
  4. If the above repetitive process halts, and no a's and b's left, then the string is accepted. Otherwise, the string is rejected.

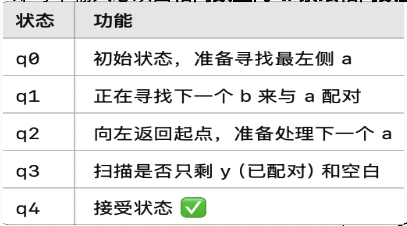


1. **找到最左边的 a**：
   - 用 X 替换它（作为 “处理过” 的标记）
2. **往右找到最左边的 b**：
   - 将其替换为 Y
3. **返回 tape 起始处**，重复步骤 1 和 2
4. **终止检查**：
   - 如果最终只剩 Xs 和 Ys，**接受**
   - 如果过程中：
     - 找不到对应 b（a 比 b 多），或
     - 找不到对应 a（b 比 a 多）→ **拒绝**

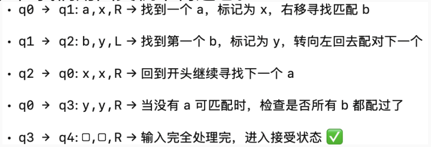
Example: Languages Accepted by Turing Machines

- $A^nB^n (n \geq 1)$: which is not regular, but context-free.

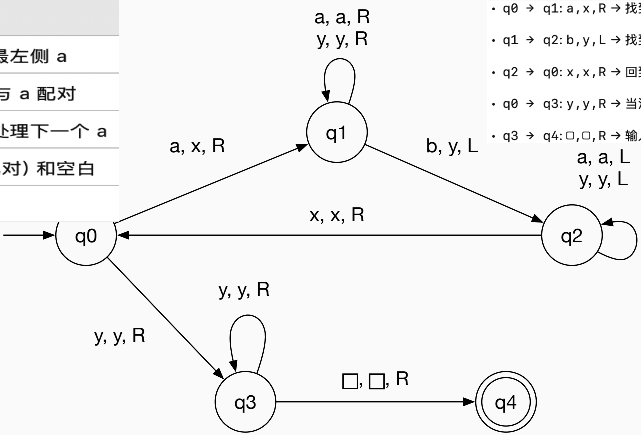
即每个输入必须由**相同数量的 a 紧跟相同数量的 b 组成**，这是典型的上下文无关语言，但我们用图灵机来构造它。



状态	功能
q0	初始状态，准备寻找最左侧 a
q1	正在寻找下一个 b 来与 a 配对
q2	向左返回起点，准备处理下一个 a
q3	扫描是否只剩 y（已配对）和空白
q4	接受状态 ✅



- q0 → q1: a,x,R → 找到一个 a，标记为 x，右移寻找匹配 b
- q1 → q2: b,y,L → 找到第一个 b，标记为 y，转向左回去配对下一个
- q2 → q0: x,x,R → 回到开头继续寻找下一个 a
- q0 → q3: y,y,R → 当没有 a 可匹配时，检查是否所有 b 都配过了
- q3 → q4: □,□,R → 输入完全处理完，进入接受状态 ✅



Example: Languages Accepted by Turing Machines

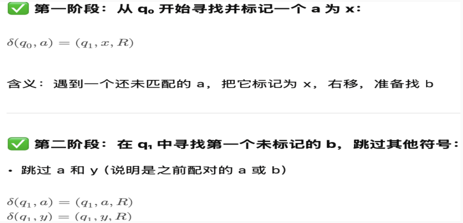
- $A^n B^n (n \geq 1)$: which is not regular, but context-free.
- $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, x, y, \square\}, \delta, q_0, \square, \{Q_4\})$ where δ is defined as follow:

  Start from q_0, replace the leftmost a with a symbol x

$$\delta(q_0, a) = (q_1, x, R)$$

  When the TM is in q_1, move right to find the leftmost b, ignore all other symbols (i.e., a and y):

$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, y) = (q_1, y, R)$$


✅ **第一阶段：从 q_0 开始寻找并标记一个 a 为 x：**

$$\delta(q_0, a) = (q_1, x, R)$$

含义：遇到一个还未匹配的 a，把它标记为 x，右移，准备找 b

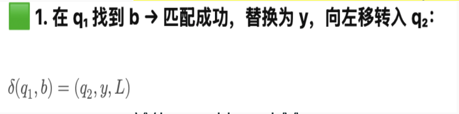
✅ **第二阶段：在 q_1 中寻找第一个未标记的 b，跳过其他符号：**

- 跳过 a 和 y（说明是之前配对的 a 或 b）

$$\delta(q_1, a) = (q_1, a, R)$$
$$\delta(q_1, y) = (q_1, y, R)$$

Example: Languages Accepted by Turing Machines

- Find the leftmost *b*, replace it with a symbol *y*, and move one step left.

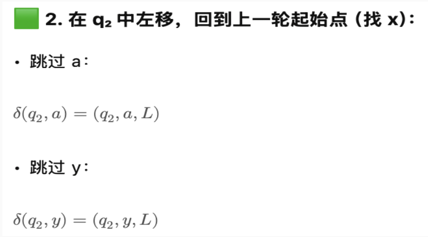


🟩 **1. 在 q₁ 找到 b → 匹配成功，替换为 y，向左移转入 q₂：**

$\delta(q_1, b) = (q_2, y, L)$

$$\delta(q_1, b) = (q_2, y, L)$$

When the TM is in q_2, move left to find the leftmost a, ignore all other symbols (i.e., a and y) until an x is found:



🟩 **2. 在 q₂ 中左移，回到上一轮起始点（找 x）：**

- **跳过 a：**

$\delta(q_2, a) = (q_2, a, L)$

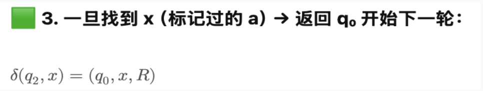
- **跳过 y：**

$\delta(q_2, y) = (q_2, y, L)$

$$\delta(q_2, a) = (q_2, a, L)$$

$$\delta(q_2, y) = (q_2, y, L)$$

We go back to q_0 once the first x is detected



🟩 **3. 一旦找到 x（标记过的 a）→ 返回 q₀ 开始下一轮：**

$\delta(q_2, x) = (q_0, x, R)$

$$\delta(q_2, x) = (q_0, x, R)$$

Example: Languages Accepted by Turing Machines

- The repeated process stops when there are no a's and b's, e.g., $xxxxx \dots xxxxxq_0yyyyy \dots yyyyy$

$$\delta(q_0, y) = (q_3, y, R)$$

$$\delta(q_3, y) = (q_3, y, R)$$

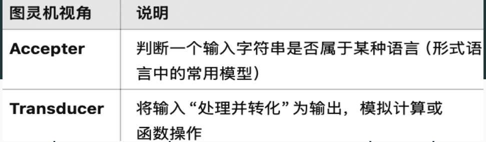
$$\delta(q_3, \square) = (q_3, \square, R)$$

Example: Languages Accepted by Turing Machines

- $A^n B^n C^n (n \geq 1)$: which is not context-free.
- This question can be solved easily using the previous idea, however, the actual program is tedious.

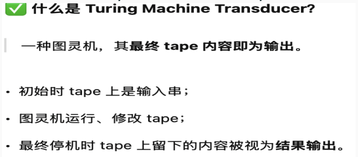
Turing Machines as Transducers

Turing Machines as Transducers



图灵机视角	说明
Accepter	判断一个输入字符串是否属于某种语言（形式语言中的常用模型）
Transducer	将输入“处理并转化”为输出，模拟计算或函数操作

- So far, we have focused on machines as language accepters, because in formal languages, accepters are quite adequate.
- On the other hand, the primary purpose of a computer is to transform input into output, i.e., it acts as a transducer.
- If we want to model computers using Turing machines, we must look at this aspect more closely.
- A *Turing machine transducer* implements a function that treats the original contents of the tape as its input and the final contents of the tape as its output.



✅ **什么是 Turing Machine Transducer?**

> 一种图灵机，其**最终 tape 内容即为输出**。

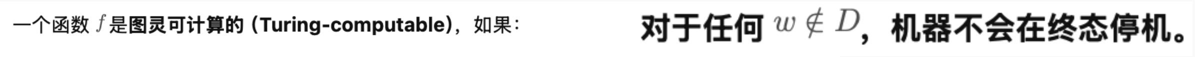
- 初始时 tape 上是输入串；
- 图灵机运行、修改 tape；
- 最终停机时 tape 上留下的内容被视为**结果输出**。

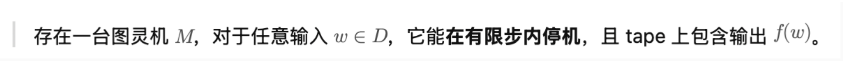
Turing Machines as Transducers

- A function is *Turing-computable* if it can be carried out by a Turing machine capable of processing all values in the function domain.
- **Definition:** A function f with domain D is said to be Turing-computable if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that, for all $w \in D$:

$$q_0 w \vdash^* q_f f(w)$$

for any input $w \notin D$, the machine M does not halt in a final state.


一个函数 f **是图灵可计算的（Turing-computable）**，如果： **对于任何 $w \notin D$，机器不会在终态停机。**


存在一台图灵机 M，对于任意输入 $w \in D$，它能**在有限步内停机**，且 tape 上包含输出 $f(w)$。

Turing Machines as Transducers

- Turing machines are the most powerful model of computation as transducers as well.

所有常见的数学函数都是图灵可计算的（Turing-computable）

- In fact, all the common mathematical functions are Turing-computable, e.g.:
    1. Arithmetic operators, Exponentiation, Integer logarithm;
    2. Comparison;
    3. String manipulation;
    4. Etc.

Example: Turing Machine as Transducers

- Given two positive integers x and y in unary notation, i.e., $\{1\}^+$, separated by a single zero, the Turing machine below computes $x + y$ which ends with a single zero:
  - For example, 111111011 means $6 + 2$, and the result should be 111111110
  - The transducer $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, \Box\}, \delta, q_0, \Box, \{q_4\})$ where δ is as follows:

$$\begin{aligned}
\delta(q_0, 1) &= (q_0, 1, R), \quad \delta(q_0, 0) = (q_1, 1, R)\\
\delta(q_1, 1) &= (q_1, 1, R), \quad \delta(q_1, \Box) = (q_2, \Box, L)\\
\delta(q_2, 1) &= (q_3, 0, L), \quad \delta(q_3, 1) = (q_3, 1, L)\\
\delta(q_3, \Box) &= (q_4, \Box, R)
\end{aligned}$$

- Derive the result for the input string “111011” by using the instantaneous description.

Example: Turing Machine as Transducers

- Derive the result for the input string “111011” by using the instantaneous description:

$$\begin{aligned} q_0111011 &\vdash 1q_011011 \vdash 11q_01011 \vdash 111q_0011 \vdash 1111q_111 \\ &\vdash 11111q_11 \vdash 111111q_1\square \vdash 11111q_21 \vdash 1111q_310 \\ &\vdash 111q_3110 \vdash 11q_31110 \vdash 1q_311110 \vdash q_3111110 \\ &\vdash q_3\square111110 \vdash q_4111110 \end{aligned}$$

Example: Turing Machine as Transducers

- Design a Turing machine that copies strings of 1′s. More precisely, find a machine that performs the computation

$$q_0 w \vdash q_f ww$$

  for any $w \in \{1\}^*$

- To solve the problem, we implement the following intuitive process:
  1. Replace every 1 by an x.
  2. Find the rightmost x and replace it with 1.
  3. Travel to the right end of the current nonblank region and create a 1 there.
  4. Repeat Steps 2 and 3 until there are no more x′s.

1. 将每个 1 替换为一个 x（表示“已处理”）
2. 找到最右的 x并复制一个 1到右端空白处
3. 回到最左，继续处理下一个 x
4. 重复直到没有 x完成复制

- **Practice:** Implement a TM to solve this problem.

Combining Turing Machines

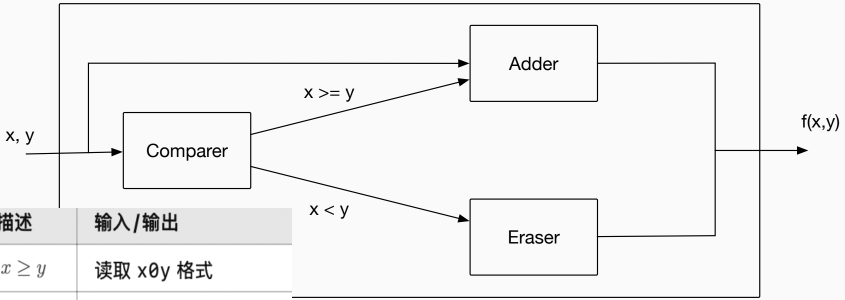
- By combining Turing Machines that perform simple tasks, complex algorithms can be implemented.
- For example, assume the existence of:
  1. a Turing machine to compare two numbers (comparer),
  2. one to add two numbers (adder),
  3. and one to erase the input (eraser)
- Design a Turing machine that computes the function:

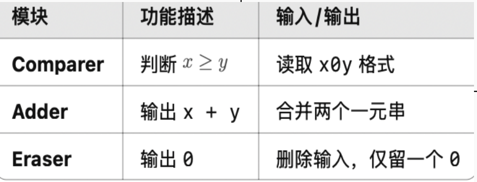
$$f(x,y) = \begin{cases} x+y & \text{if} \quad x \geq y \\ 0 & \text{if} \quad x < y \end{cases}$$

Combining Turing Machines

- Design a Turing machine that computes the function:

$$f(x,y) = \begin{cases} x+y & \text{if} \quad x \geq y \\ 0 & \text{if} \quad x < y \end{cases}$$



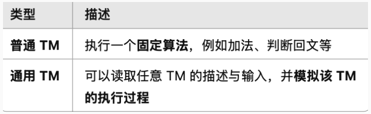


模块	功能描述	输入/输出
Comparer	判断 $x \geq y$	读取 x0y 格式
Adder	输出 x + y	合并两个一元串
Eraser	输出 0	删除输入，仅留一个 0

Universal Turing Machines

- The TMs that we have studied so far have been special-purpose computers capable of executing a single algorithm.

- We can consider a “universal” Turing machine, which can execute a program stored in its memory:
- It receives an input string that specifies both:
  1. the algorithm it is to execute
  2. and the input that is to be provided to the algorithm.



类型	描述
普通 TM	执行一个**固定算法**，例如加法、判断回文等
通用 TM	可以读取任意 TM 的描述与输入，并**模拟该 TM 的执行过程**

- Think of your personal computer:
  1. Instead of buying a separate computer for each software, we would rather have one machine that can run any arbitrary software.
  2. We encode the software in 0s and 1s (i.e., binary format) and provide it to our machines.

The Church-Turing Thesis

The Church-Turing Thesis

- To say that the TM is a general model of computation implies that:
    - “any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM”

任何可以被执行的算法性过程，都可以被某台图灵机执行。如果某个任务是 “ 可计算的 ” ，那么图灵机就一定能够实现它

- The statement was formulated by Alonzo Church in the 30’s.
    - It is referred to as Church’s thesis or the Church-Turing thesis.
    - **It isn’t a statement that can be proved**.
    - But there is a lot of evidence for it.

- An acceptance of the Church-Turing Thesis leads to a definition of an algorithm:
    - An algorithm for a function $f : D \to R$ is a Turing machine M, which given any $d \in D$ on its tape, eventually halts with the correct answer $f(d) \in R$ on its tape.

The Church-Turing Thesis

- The nature of the model makes it seem that a *TM* can execute any algorithm a human can. **图灵机能执行人类能执行的所有 “ 算法 ”**
- Apparent enhancements to the TM have been shown not to increase its power. **对图灵机的 “ 表面增强 ” 并不增加其计算能力**
- Other models of computation proposed have either been less powerful or equivalent to Turing machines. **其他计算模型都与图灵机等价或更弱**

The Church-Turing Thesis

至今没有任何被广泛接受的 “ 有效计算 ” 模型超出图灵机的能力。

- No one has ever suggested any kind of **effective** computation that cannot be implemented on a TM:
  - Pay attention to the word “effective”.
  - For those who are interested, see (e.g.):
    - https://en.wikipedia.org/wiki/Hypercomputation
    - Hava T. Siegelmann, “Neural and Super-Turing Computing”, Minds and Machines 13: 103–114, 2003.

- From now on, we will consider that, by definition, an “algorithmic procedure” is what a Turing machine can do.

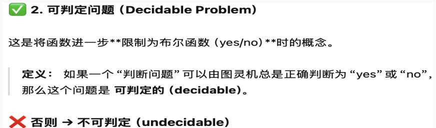
Computability and Decidability

- **Definition:** a function *f* on a certain domain is said to be computable if there exists a Turing machine that computes the value of *f* for all arguments in its domain.

若不存在这样的图灵机 函数是不可计算（uncomputable）

- A function is uncomputable if no such Turing machine exists.
- If we simplify the problem to “yes” or “no” questions, then we talk about a problem being **decidable** or **undecidable**.

- We say that a problem is **decidable** if there exists a Turing machine that gives the correct answer for every statement in the domain of the problem.

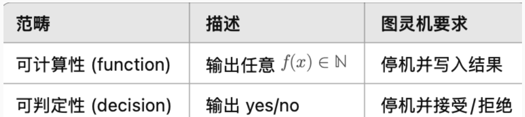


✅ **2. 可判定问题（Decidable Problem）**

这是将函数进一步**限制为布尔函数（yes/no）**时的概念。

> **定义：** 如果一个“判断问题”可以由图灵机总是正确判断为“yes”或“no”，那么这个问题是 **可判定的（decidable）**。

❌ **否则 → 不可判定（undecidable）**



范畴	描述	图灵机要求
可计算性 (function)	输出任意 $f(x) \in \mathbb{N}$	停机并写入结果
可判定性 (decision)	输出 yes/no	停机并接受/拒绝

The Turing Machine Halting Problem

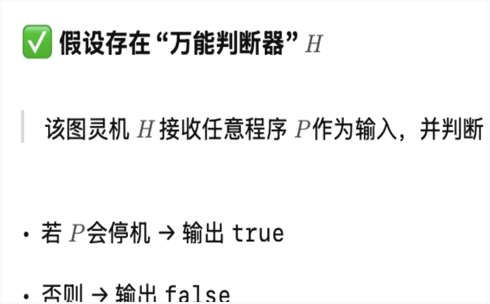
没有一台图灵机能够总是正确地判断任意图灵机 M 在输入 w 上是否会停机。

- Given the description of a Turing machine M and an input w, does M, when started in the initial configuration, perform a computation that eventually halts?
- This is a famous problem that cannot be solved by TM.
- How to prove it?

The Turing Machine Halting Problem

Proof by contradiction:
Assume we have a procedure (TM) *H* which takes a program (TM) *P* as its input and answers true if *P* halts on all its possible input and false otherwise. Given the procedure *H*, we could construct another program *H*′ such that:

```
Program H'()
      If H(H') then
            Loop
      else
            Halt;
```

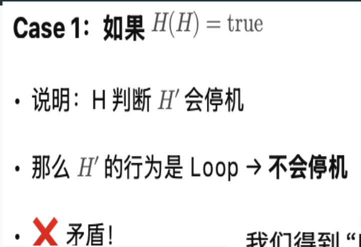

✅ **假设存在“万能判断器”** H

> 该图灵机 H 接收任意程序 P 作为输入，并判断

- 若 P 会停机 → 输出 `true`
- 否则 → 输出 `false`

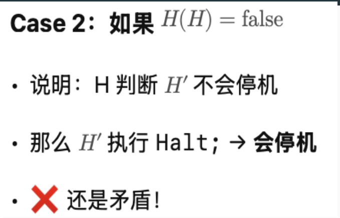
The Turing Machine Halting Problem

```
Program H'()
      If H(H') then
            Loop
      else
            Halt;
```



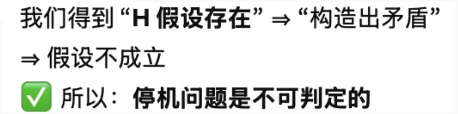
Case 1：如果 $H(H) = \text{true}$

- 说明：H 判断 H' 会停机
- 那么 H' 的行为是 Loop → **不会停机**
- ❌ 矛盾！



Case 2：如果 $H(H) = \text{false}$

- 说明：H 判断 H' 不会停机
- 那么 H' 执行 `Halt;` → **会停机**
- ❌ 还是矛盾！



我们得到“**H 假设存在**” ⇒ “构造出矛盾”

⇒ 假设不成立

✅ 所以：**停机问题是不可判定的**

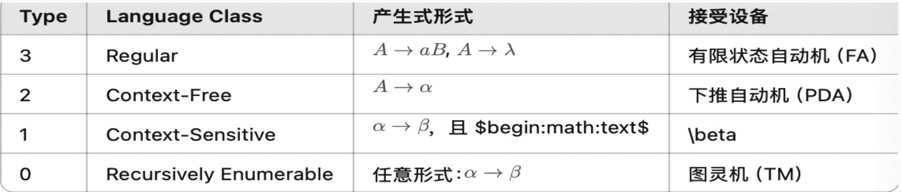
If H' will terminate on all its input, *i.e.*, $H(H')$ returns true, then H' enters an infinite loop. If H' will not terminate, then $H(H')$ returns false, which will then lead H' to a halting state. Contradiction.

The Chomsky Hierarchy 乔姆斯基层级

$\text{Regular} \subset \text{Context}-\text{Free} \subset \text{Context}-\text{Sensitive} \subset \text{RE}\ (\text{Type}-0)$

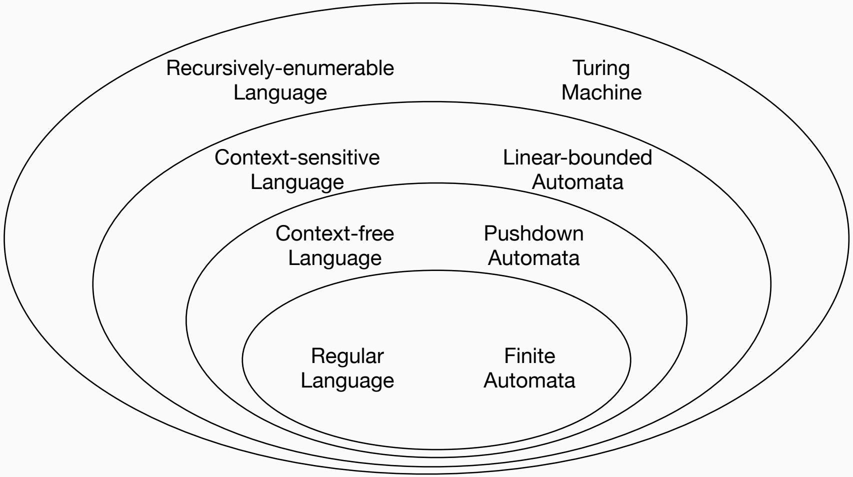
Type	Languages	Form of productions	Accepting Device
3	Regular	$A \to aB$, $A \to \lambda$	Finite Automata
2	Context-Free	$A \to \alpha$	Pushdown Automata
1	Context-Sensitive	$\alpha \to \beta$, with $\|\beta\| \geq \|\alpha\|$	Linear Bounded Automata
0	Unrestricted	$\alpha \to \beta$	Turing Machine

Table 1: The Chomsky Hierarchy



Type	Language Class	产生式形式	接受设备
3	Regular	$A \to aB$, $A \to \lambda$	有限状态自动机（FA）
2	Context-Free	$A \to \alpha$	下推自动机（PDA）
1	Context-Sensitive	$\alpha \to \beta$, 且 $begin:math:text$	\beta
0	Recursively Enumerable	任意形式：$\alpha \to \beta$	图灵机（TM）

The Chomsky Hierarchy


Recursively-enumerable Language
Turing Machine
Context-sensitive Language
Linear-bounded Automata
Context-free Language
Pushdown Automata
Regular Language
Finite Automata