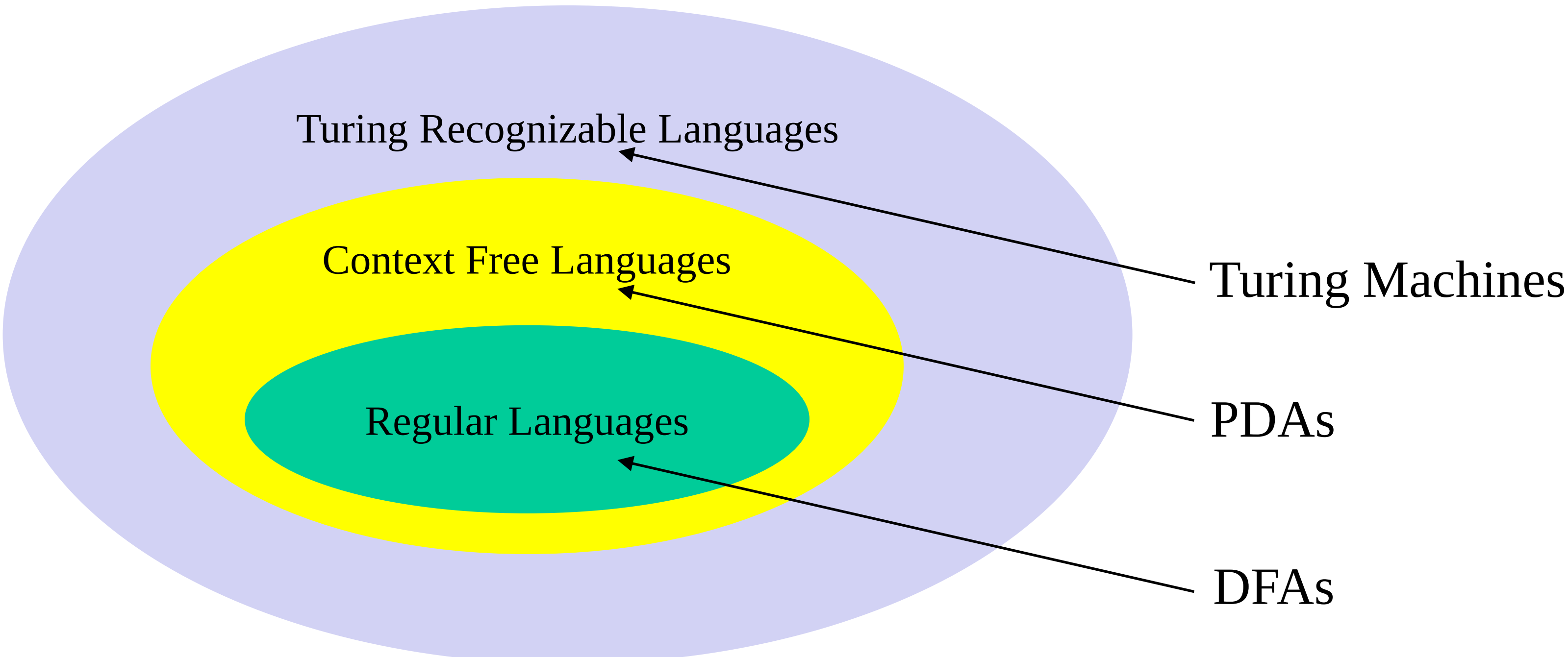
Turing Machines

Turing Machines

- DFAs recognize regular languages.
- PDAs recognize CFLs.
- There are languages that are NOT CFLs.



Turing Machines

Regular Language, CFL, Non-CFL

- Identify the following languages as **regular language**, **CFL** or **non-CFL**.

- $\{ 0^n 1^m \mid n \geq 0 \text{ and } m \geq 0 \}$
- $\{ 0^n 1^n \mid n \geq 0 \}$
- $\{ 0^n 1^m 0^n \mid n \geq 0 \text{ and } m \geq 0 \}$
- $\{ 0^n 1^n 0^n \mid n \geq 0 \}$
- $\{ 0^n 1^{2n} \mid n \geq 0 \}$
- $\{ 0^n 1^{2m} \mid n \geq 0 \text{ and } m \geq 0 \}$
- $\{ 0^n 1^n 2^n \mid n \geq 0 \}$
- $\{ 0^i 1^j 2^k \mid k \geq j \geq i \geq 0 \}$
- $\{ w\#w \mid w \text{ is in } \{0,1\}^* \}$

Turing Machines

Regular Language, CFL, Non-CFL

- Identify the following languages as **regular language**, **CFL** or **non-CFL**.

- $\{ 0^n1^m \mid n \geq 0 \text{ and } m \geq 0 \}$ **regular language**
- $\{ 0^n1^n \mid n \geq 0 \}$ **CFL**
- $\{ 0^n1^m0^n \mid n \geq 0 \text{ and } m \geq 0 \}$ **CFL**
- $\{ 0^n1^n0^n \mid n \geq 0 \}$ **non-CFL**
- $\{ 0^n1^{2n} \mid n \geq 0 \}$ **CFL**
- $\{ 0^n1^{2m} \mid n \geq 0 \text{ and } m \geq 0 \}$ **regular language**
- $\{ 0^n1^n2^n \mid n \geq 0 \}$ **non-CFL**
- $\{ 0^i1^j2^k \mid k \geq j \geq i \geq 0 \}$ **non-CFL**
- $\{ w\#w \mid w \text{ is in } \{0,1\}^* \}$ **non-CFL**

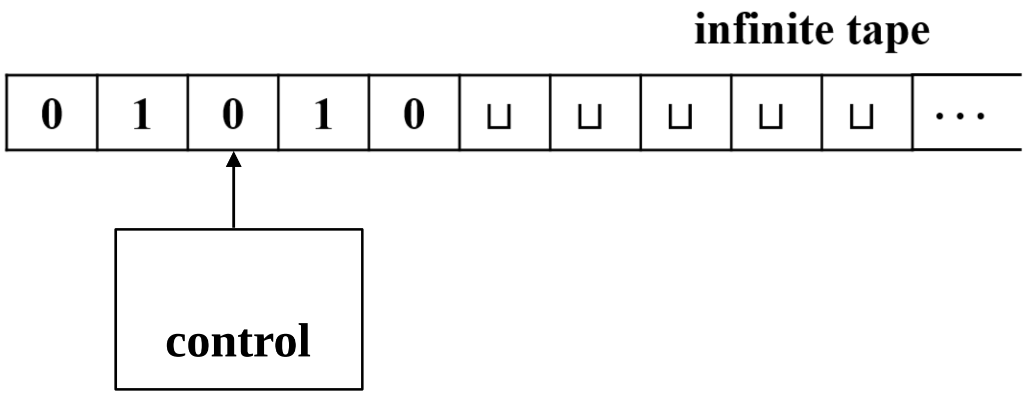
Turing Machines

- There is a much more powerful model of computation called **Turing Machines (TM).**
  - It is first proposed by Alan Turing in 1936

Alan Turing
1912-1954

- TMs are similar to a finite automaton, but a TM has an **unlimited memory**.
- A Turing machine is a much more accurate model of a general purpose computer.
- **A Turing machine can do everything that a real computer can do**.
- **Even a TM can not solve certain problems.**
  - Such problems are beyond theoretical limits of computation.

Turing Machines



- The Turing machine model uses an **infinite tape** as its unlimited memory.
- It has a **tape head** that can read and write symbols on the tape.
- Tape head can move to Left or Right.

Turing Machine Computation

- Read/write head starts at leftmost position on tape
- Input string written on leftmost squares of tape, rest is **blank**
- Computation proceeds according to **transition function**:
  - Given ***current state of machine***, and ***current symbol being read***
  - the machine
    - transitions to new state
    - writes a symbol to its current position (overwriting existing symbol)
    - moves the tape head L or R
- Computation ends if and when it enters either the **accept** or the **reject** state.

How does a TM Compute?

- Consider the language $L = \{ w\#w \mid w \in \{0,1\}^* \}$
- We already know that L is not a regular language and it is not a CFL.
- But there is a TM that recognizes L.

Idea for Turing machine

- Zig-zag across tape to corresponding positions on either side of **'#'** to check whether these positions agree.
  - If they do not, or if there is no **'#'**, ***reject***.
  - If they do, cross them off.
- Once all symbols to the left of the **'#'** are crossed off, check for any symbols to the right of **'#'**:
  - if there are any, ***reject***;
  - if there aren't, ***accept***.

How does a TM Compute?

- Consider the language L = { w#w | w ∈{0,1}* }
- The TM starts with the input on the tape.

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ ⊔ ⊔ cross the symbol, move to R

X 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ ⊔ ⊔

→→ . . . move to R until first un-crossed symbol after #, it must be 0

X 1 1 0 0 0 # X 1 1 0 0 0 ⊔ ⊔ ⊔

←← . . . move to L until first crossed symbol after #

X 1 1 0 0 0 # X 1 1 0 0 0 ⊔ ⊔ ⊔ move to R

X X 1 0 0 0 # X 1 1 0 0 0 ⊔ ⊔ ⊔

→→ . . . move to R until first un-crossed symbol after #, it must be 1

⋮ Repeat ziz-zag operations

X X X X X X # X X X X X X ⊔ ⊔ ⊔ move to R

X X X X X X # X X X X X X ⊔ ⊔ ⊔

→→ . . . move to R until first un-crossed symbol, it must be blank

X X X X X X # X X X X X X ⊔ ⊔ ⊔ **ACCEPT**

Formal Definition of a Turing Machine

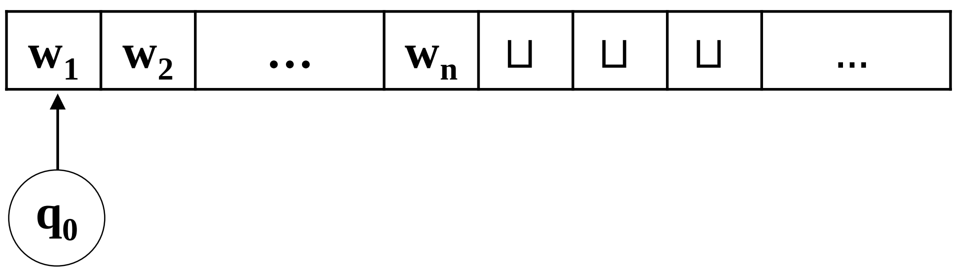
A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where Q, Σ, Γ are all finite sets and

1. **Q is the set of states,**
2. **Σ is the input alphabet not containing the *blank symbol* $\sqcup$,**
3. **Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$,**
4. **δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is the transition function,**
5. **$q_0 \in Q$ is the start state,**
6. **$q_{accept} \in Q$ is the accept state, and**
7. **$q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$**

How does a Turing Machine Compute?

Initial Configuration:

- A Turing Machine M receives its input $w = w_1 w_2 \ldots\ w_n$ on the leftmost **n** squares on the tape. The rest of the tape is blank.
  - The head starts on the leftmost square on the tape.
  - The first blank symbol on the tape marks the end of the input.
- The initial state is the start state $\mathbf{q_0}$.



How does a Turing Machine Compute?

Transition:

- The computation proceeds according to the **transition function δ**.
  - If the current state is $\mathbf{q_i}$, the current tape symbol **a**, and $\boldsymbol{\delta(q_i,a)=(q_j,b,D)}$ where **a** and **b** are tape symbols (they can be the same symbol) and **D** is **L** or **R**, then
    - Machine M goes from the state $\mathbf{q_i}$ to state $\mathbf{q_j}$.
    - Machine M writes **b** onto the current tape position (if **a** and **b** are same, no change occurs on the current tape position).
    - Tape head moves to **Left** (if **D** is **L**) or moves to **Right** (if **D** is **R**)

- The head of Machine M never moves left of the beginning of the tape.
  - **If Machine M is on the leftmost square, it stays there!**

How does a Turing Machine Compute?

Accepting or Rejecting:

- The computation proceeds until Machine M enters either $\mathbf{q_{accept}}$ or $\mathbf{q_{reject}}$, **when it halts**.
  - If Machine M enters $\mathbf{q_{accept}}$ ➔ **ACCEPT**
  - If Machine M enters $\mathbf{q_{reject}}$ ➔ **REJECT**

- **The Turing Machine M may go on forever, never halting!**

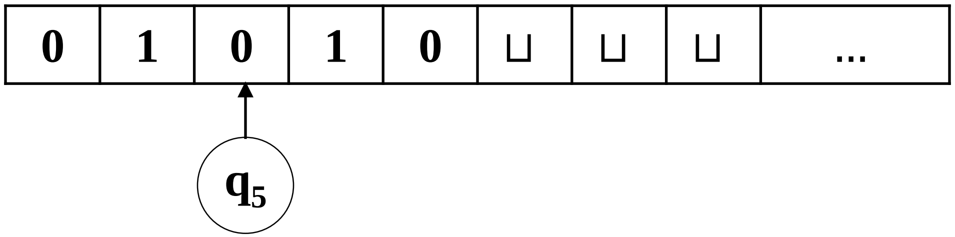
Configuration of a Turing Machine

- As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location.
- **Each step of a TM computation can be captured by the notion of a configuration.**

Configuration:

- **For a state q and two strings u and v over the tape alphabet Γ, we write uqv for the configuration where the current state is q, the current tape contents is uv, and the current head location is the first symbol of v.**
- **The tape contains only blanks following the last symbol of v.**

Configuration of a Turing Machine



- The current state is q_5,
- **u = 01** is **to the left of the head**,
- **v = 010** is **under and to the right of the head**.
- Tape has **uv = 01010** on it.

- **We represent this configuration by $01q_5010$**

Configurations

- **Configuration C_1 yields ($\Rightarrow$) configuration C_2 if TM can legally go from C_1 to C_2 in a single step.**
- $ua\ q_i\ bv \Rightarrow u\ q_j\ acv$ if $\delta(q_i,b) = (q_j,c,L)$
- $ua\ q_i\ bv \Rightarrow uac\ q_j\ v$ if $\delta(q_i,b) = (q_j,c,R)$
- $q_i\ bv \Rightarrow q_j\ cv$ if $\delta(q_i,b) = (q_j,c,L)$
  - If head is at the left end and the transition is left-moving, then head stay at the left end.
- $q_i\ bv \Rightarrow c\ q_j\ v$ if $\delta(q_i,b) = (q_j,c,R)$
  - If the head is at the left end, and the transition is right-moving.
- $ua\ q_i \Rightarrow uac\ q_j$ if $\delta(q_i,\sqcup) = (q_j,c,R)$
  - If the head is at the right end, configuration **$ua\ q_i$** is equivalent to configuration **$ua\ q_i\ \sqcup$** because we assume that blanks follow the part of the tape represented in the configuration.
- $q_i \Rightarrow c\ q_j$ if $\delta(q_i,\sqcup) = (q_j,c,R)$
- $q_i \Rightarrow q_j\ c$ if $\delta(q_i,\sqcup) = (q_j,c,L)$
  - If the head is at both left and right ends, configuration **q_i** is equivalent to configuration **$q_i\ \sqcup$.**

Configurations

- The **start configuration** of a TM on input **w** is the **configuration q_0 w**, which indicates that the machine is in the **start state q_0** with its head at the leftmost position on the tape.
- In an **accepting configuration**, the state of the configuration is q_{accept}.
- In a **rejecting configuration**, the state of the configuration is q_{reject}.
- **Accepting and rejecting configurations** are **halting configurations** and do not yield further configurations.

- **Because the machine is defined to halt when in the states q_{accept} and q_{reject},** we equivalently could have defined the transition function to as follows:

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

where Q' is Q without q_{accept} and q_{reject}

Language of a Turing Machine

Accepting (Recognizing) String:

- A Turing machine M accepts an input string w if a sequence of configurations C_1, C_2, . . . , C_k exists, where
  1. C_1 is the start configuration of M on input w,
  2. Each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration.

Language of a Turing Machine M (or Language Recognized by M):

- The language of A Turing machine M L(M) is the set of strings w that are recognized by M.
- More formally,

  L(M) = { w | q_0 w $\Rightarrow^*$ C where q_0 w is the starting configuration of M on input w and C is an accepting configuration of M. }

Language of a Turing Machine

Turing-Recognizable:

- **A language L is Turing-recognizable if some Turing machine recognizes it.**
  - **It is also called a recursively enumerable language.**

- When we start a Turing machine on an input, three outcomes are possible.
  - The machine may ***accept***, ***reject***, or ***loop***.
  - By ***loop*** we mean that **the machine simply does not halt**.
  - **Looping** may entail any simple or complex behavior that never leads to a halting state.

Language of a Turing Machine

- A ***Turing machine M can fail to accept*** an input by ***entering q_{reject} state*** or ***looping***.
  - We prefer Turing machines that halt on all inputs; such machines never loop.
  - These machines are called deciders because they always make a decision to accept or reject.
  - **A decider that recognizes some language also is said to decide that language.**

Turing-Decidable (Decidable):

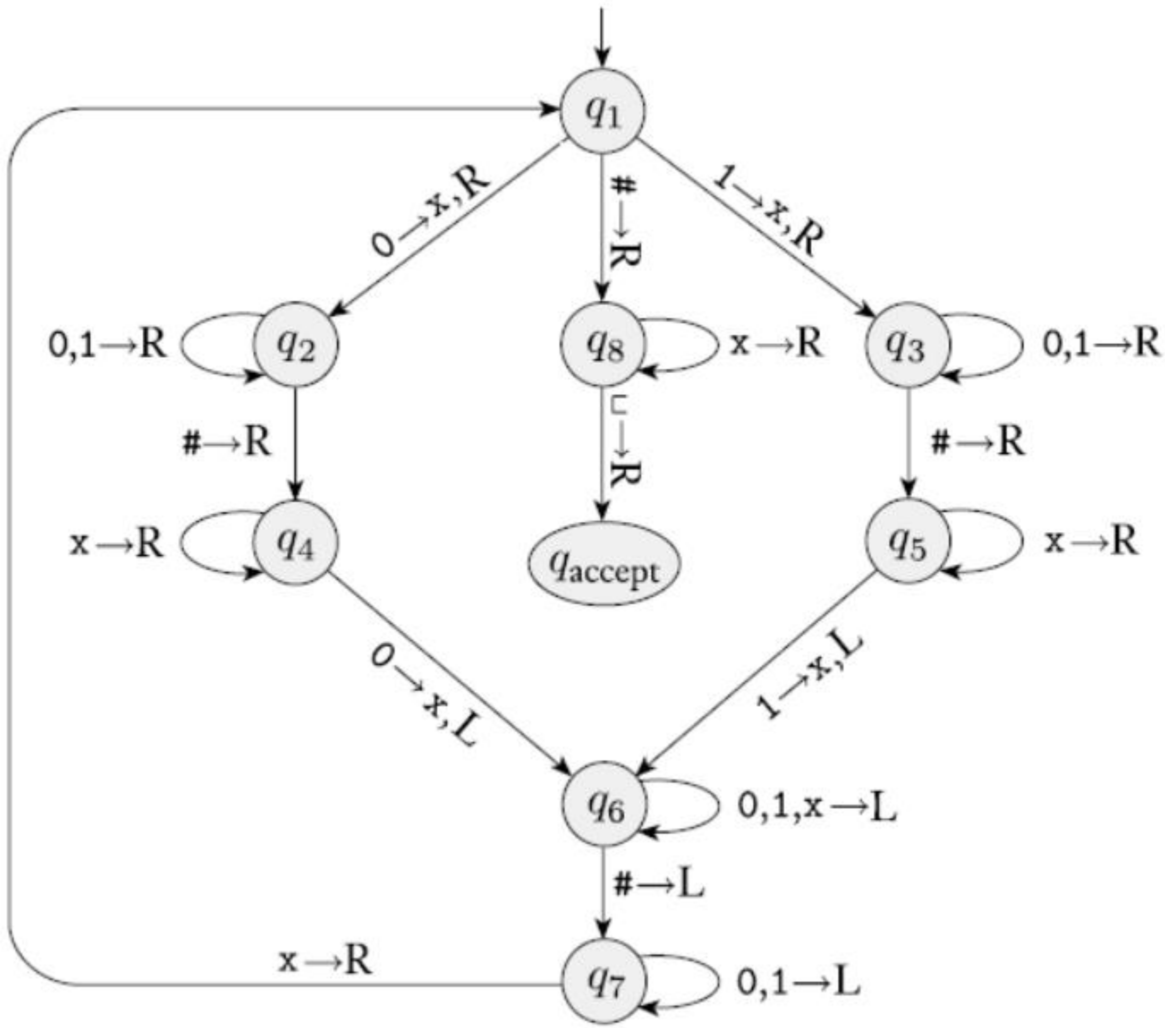
- **A language L is Turing-decidable (or decidable) if some Turing machine (which is a decider) decides it.**
  - **It is also called a recursive language.**

- **Every decidable language is Turing-recognizable.**
  - **But there are languages that are Turing-recognizable but not decidable.**

Example: Turing Machine

- Consider $L = \{ w\#w \mid w \in \{0,1\}^* \}$
- A formal description of a TM which decides L is $(Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ where
  - $Q = \{q_1, \ldots, q_8, q_{accept}, q_{reject}\}$
  - $\Sigma = \{0,1,\#\}$
  - $\Gamma = \{0,1,\#,x,\sqcup\}$
  - The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.
  - We describe δ with a state diagram.
    - On the given state diagram, q_{reject} is missing and some transitions are also missing.
    - We assume that all missing transitions goes to state q_{reject} without changing the current tape symbol and head moves to right.

Example: Turing Machine



- q_{reject} is missing and all missing transitions goes to state q_{reject} .

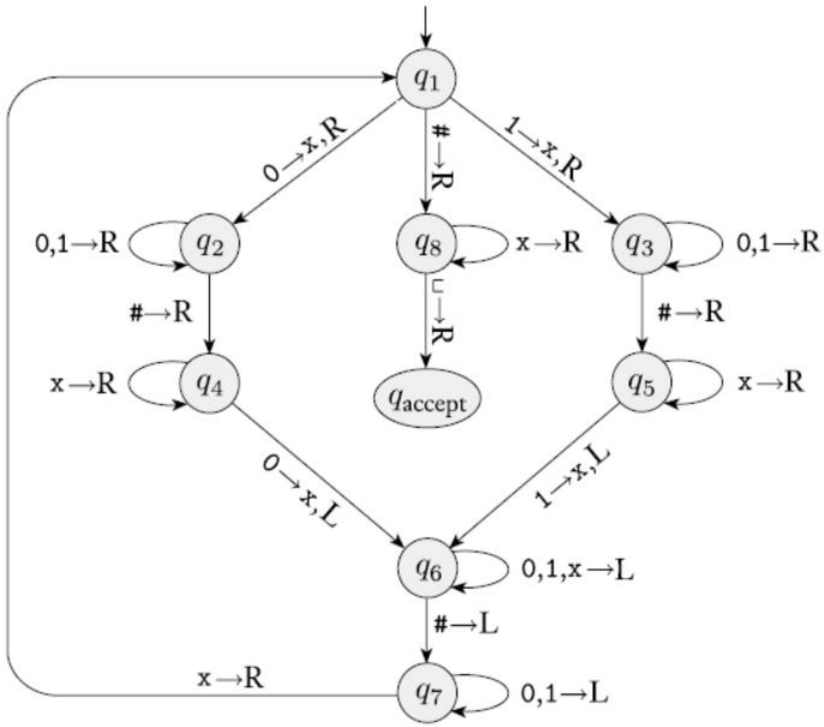
Arc label meanings:

- $0 \to x,R$ from q_1 to q_2 means: $\delta(q_1,0)=(q_2,x,R)$
- $\# \to R$ from q_1 to q_8 means: $\delta(q_1,\#)=(q_8,\#,R)$
- $0,1 \to R$ from q_3 to q_3 means: $\delta(q_3,0)=(q_3,0,R)$ and $\delta(q_3,1)=(q_3,1,R)$
- $0,1,x \to L$ from q_6 to q_6 means: $\delta(q_6,0)=(q_6,0,L)$ $\delta(q_6,1)=(q_6,1,L)$ and $\delta(q_6,x)=(q_6,x,L)$

Missing arcs from q_8 means:
$\delta(q_8,0)=(q_{reject},0,R)$
$\delta(q_8,1)=(q_{reject},1,R)$
$\delta(q_8,\#)=(q_{reject},\#,R)$

Example: Turing Machine

An Accepting Computation



The computation of 01#01

q_1 01#01 $\Rightarrow$ x q_2 1#01 $\Rightarrow$ x1 q_2 #01 $\Rightarrow$ x1# q_4 01

$\Rightarrow$ x1 q_6 #x1 $\Rightarrow$ x q_7 1#x1 $\Rightarrow$ q_7 x1#x1 $\Rightarrow$ x q_1 1#x1

$\Rightarrow$ xx q_3 #x1 $\Rightarrow$ xx# q_5 x1 $\Rightarrow$ xx#x q_5 1 $\Rightarrow$ xx# q_6 xx

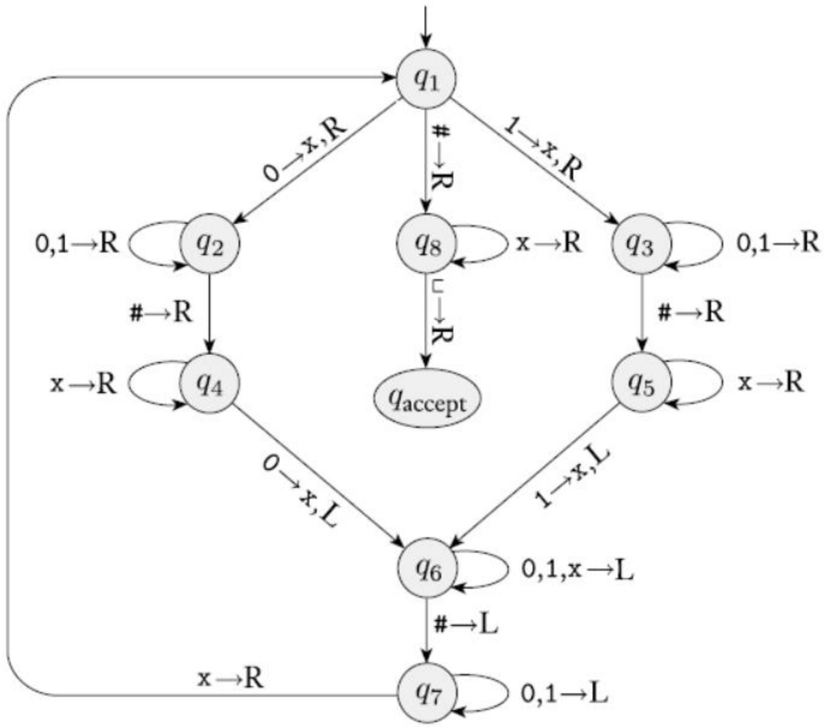
$\Rightarrow$ xx q_6 #xx $\Rightarrow$ x q_7 x#xx $\Rightarrow$ xx q_1 #xx $\Rightarrow$ xx# q_8 xx

$\Rightarrow$ xx#x q_8 x $\Rightarrow$ xx#xx q_8 $\sqcup$ $\Rightarrow$ **xx#xx**$\sqcup$ $\mathbf{q_{accept}}$ $\sqcup$

ACCEPT

Example: Turing Machine

A Rejecting Computation



The computation of 01#00

$q_1\ 01\#00 \Rightarrow x\ q_2\ 1\#00 \Rightarrow x1\ q_2\ \#00 \Rightarrow x1\#\ q_4\ 00$

$\Rightarrow x1\ q_6\ \#x0 \Rightarrow x\ q_7\ 1\#x0 \Rightarrow q_7\ x1\#x0 \Rightarrow x\ q_1\ 1\#x0$

$\Rightarrow xx\ q_3\ \#x0 \Rightarrow xx\#\ q_5\ x0 \Rightarrow xx\#x\ \mathbf{q_5\ 0}$

$\Rightarrow \mathbf{xx\#x0\ q_{reject}}\ \sqcup$

REJECT

Example 2: Turing Machine

- A Turing Machine (TM) M that **decides** L = { 0^{2^n} | n ≥ 0 } which is the language consisting of all strings of 0s whose length is a power of 2.

- TM M "On input string w:
  1. Sweep left to right across the tape, crossing off every other 0.
  2. If in stage 1 the tape contained a single 0, **accept** .
  3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, **reject** .
  4. Return the head to the left-hand end of the tape.
  5. Go to stage 1."

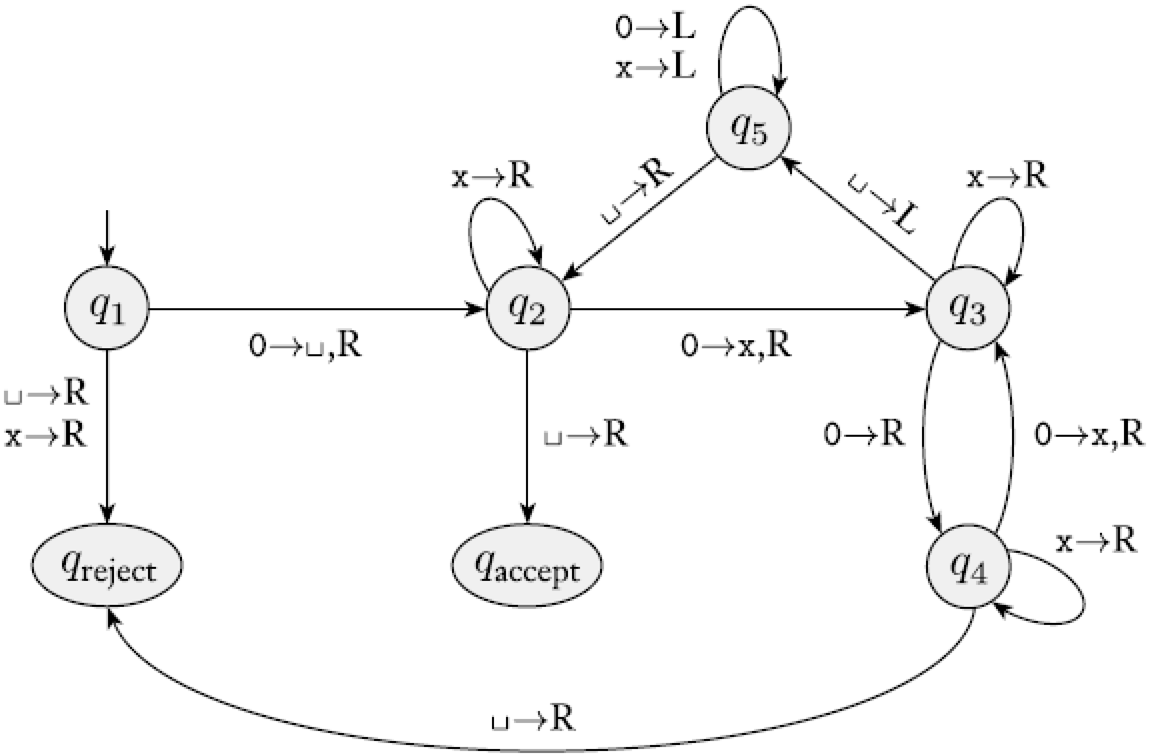
- Each iteration of stage 1, M cuts the number of 0s in half. As the machine sweeps across the tape in stage 1, it keeps track of whether the number of 0s seen is even or odd.
  – If that number is odd and greater than 1, the original number of 0s in the input could not have been a power of 2. Therefore, the machine **rejects** in this instance.
  – However, if the number of 0s seen is 1, the original number must have been a power of 2. So in this case, the machine **accepts**.

Example 2: Turing Machine

- A Turing Machine (TM) M that **decides** L = { 0^{2^n} | n ≥ 0 } which is the language consisting of all strings of 0s whose length is a power of 2.

- A formal description of a TM which decides L is (Q, Σ, Γ, δ, q_1, q_{accept}, q_{reject}) where
  - Q = {q_1,…,q_5, q_{accept}, q_{reject} }
  - Σ = {0}
  - Γ = {0,x,⊔}
  - The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.
  - We describe δ with a state diagram.

Example 2: Turing Machine



- A Turing Machine (TM) M that **decides** L = { 0^{2^n} | n ≥ 0 }.

- A TM M is

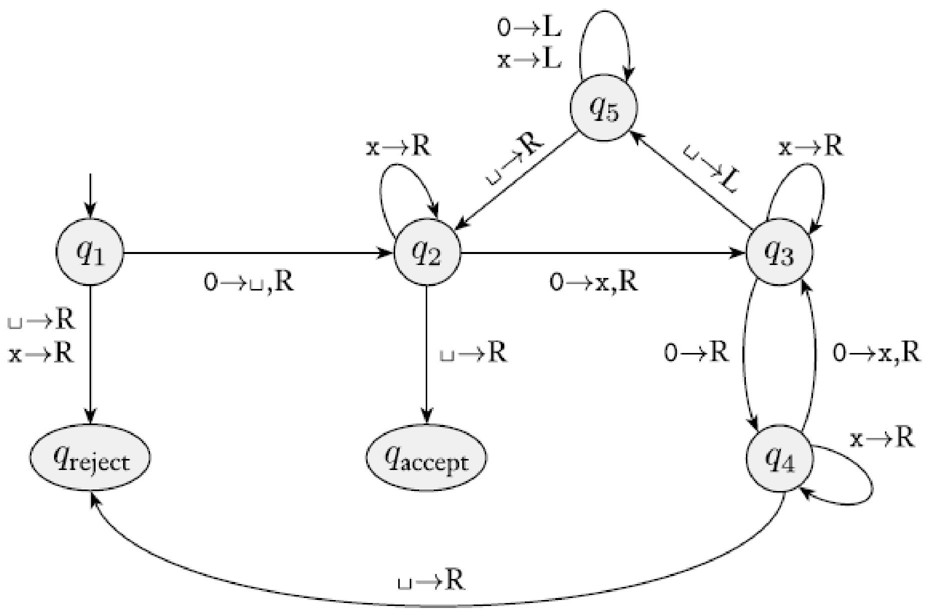
  (Q, Σ, Γ, δ, q_1, q_{accept}, q_{reject})

  – $Q = \{q_1,\ldots,q_5, q_{accept}, q_{reject}\}$

  – $\Sigma = \{0\}$

  – $\Gamma = \{0,x,\sqcup\}$

Example 2: Turing Machine



The computation of 0000

$q_1\ 0000 \Rightarrow \sqcup\ q_2\ 000 \Rightarrow \sqcup x\ q_3\ 00 \Rightarrow \sqcup x0\ q_4\ 0$

$\Rightarrow \sqcup x0x\ q_3\ \sqcup \Rightarrow \sqcup x0\ q_5\ x\sqcup \Rightarrow \sqcup x\ q_5\ 0x\sqcup$

$\Rightarrow \sqcup\ q_5\ x0x\sqcup \Rightarrow q_5\ \sqcup x0x\sqcup \Rightarrow \sqcup\ q_2\ x0x\sqcup$

$\Rightarrow \sqcup x\ q_2\ 0x\sqcup \Rightarrow \sqcup xx\ q_3\ x\sqcup \Rightarrow \sqcup xxx\ q_3\ \sqcup$

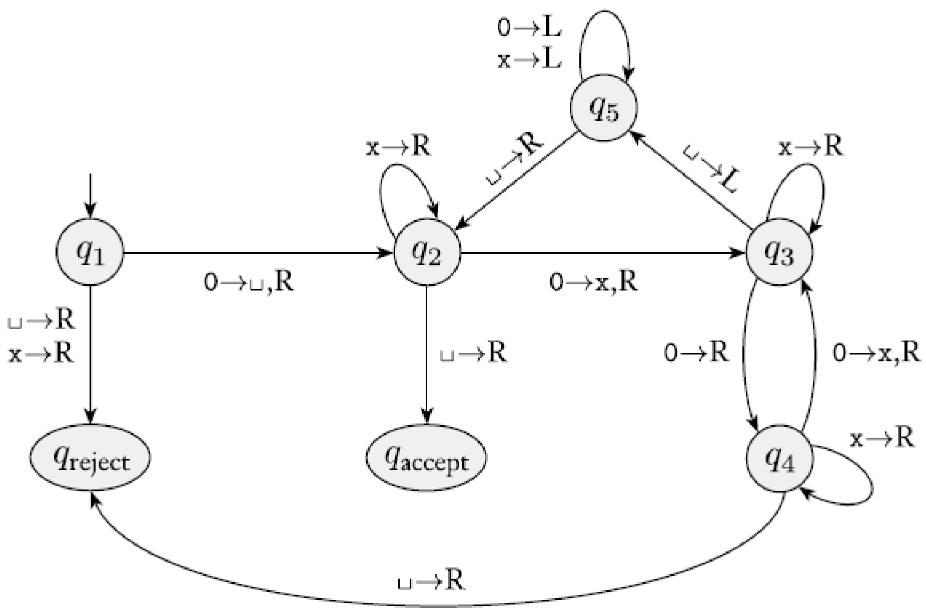
$\Rightarrow \sqcup xx\ q_5\ x\sqcup \Rightarrow \sqcup x\ q_5\ xx\sqcup \Rightarrow \sqcup\ q_5\ xxx\sqcup$

$\Rightarrow q_5\ \sqcup xxx\sqcup \Rightarrow \sqcup\ q_2\ xxx\sqcup \Rightarrow \sqcup x\ q_2\ xx\sqcup$

$\Rightarrow \sqcup xx\ q_2\ x\sqcup \Rightarrow \sqcup xxx\ q_2\ \sqcup \Rightarrow \sqcup xxx\sqcup\ q_{accept}$

ACCEPT

Example 2: Turing Machine



The computation of 000

$q_1\ 000 \Rightarrow \sqcup\ q_2\ 00 \Rightarrow \sqcup x\ q_3\ 0 \Rightarrow \sqcup x0\ q_4\ \sqcup$
$\Rightarrow \sqcup x0\sqcup\ q_{reject}$

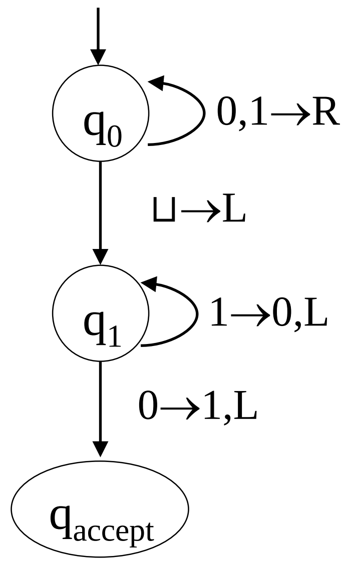
REJECT

Example 3: Turing Machine

- A TM to add 1 to a binary number (with a 0 in front)
- M = “On input w
  1. Go to the right end of the input string
  2. Move left as long as a 1 is seen, changing it to a 0.
  3. Change the 0 to a 1, and halt.”
- For example, to add 1 to w = 0110011
  - Change all the ending 1’s to 0’s ⇒ 0110000
  - Change the next 0 to a 1 ⇒ 0110100
- Now let’s design a TM for this problem.

Example 3: Turing Machine

A TM to add 1 to a binary number (with a 0 in front)



Turing Machines: Story So Far!

- **Turing Machines are the most general model of computation.**

- **Computations of a TM are described by a sequence of configurations.**
  - Accepting Configuration: contains state q_{accept}
  - Rejecting Configuration: contains state q_{reject}
  - Starting Configuration for input w: q_0 w where q_0 is the start state

- **Turing-recognizable languages**
  - TM halts in an accepting configuration if w is in the language.
  - TM may halt in a rejecting configuration or go on indefinitely if w is not in the language.

- **Turing-decidable languages**
  - TM halts in an accepting configuration if w is in the language.
  - TM halts in a rejecting configuration if w is not in the language.

Variants of Turing Machines

- We talked the standard model of Turing Machines.
- **A standard (ordinary) TM**
  - has a **single tape** and a **single read/write head** which move to Left or Right.
  - is **deterministic**.

- There alternative definitions of Turing Machines, and they are called **variants** of Turing machine model.

- **Some variants of TMs are:**
  - **Turing Machines with Stay Option**
  - **Multitape Turing Machines**
  - **Non-Deterministic Turing Machines**
  - **Enuramators**
  - …

Variants of Turing Machines
Equivalence of Power

- A computational model is **robust** if the class of languages it accepts does not change under variants.
  - We have seen that DFA's are robust for nondeterminism.
    - NFAs and DFAs accept the same class of languages.
  - But not PDAs!
    - Non-deterministic PDAs are more powerful than Deterministic PDAs

- The robustness of Turing Machines is by far greater than the **robustness** of DFAs and PDAs.

- We introduce several variants on Turing machines and show that **all these variants have equal computational power.**
  - **Each variant has the same power with Ordinary Turing Machine.**
  - **All of them accept the same set of languages (Turing-Recognizable languages).**

Variants of Turing Machines

Equivalence of Power

Same Power of two classes (variants) means:

- **For any machine M_1 of first class there is a machine M_2 of second class such that:**

  $$L(M_1) = L(M_2)$$

  and vice-versa.

Simulation:

- **In order to prove that two classes of TMs have same power, we can simulate the machine of the first class with a machine of the other class.**

Turing Machines with Stay Option

- Suppose in addition moving Left or Right, we give the option to the TM to **stay (S)** on the current cell, that is:

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

- **A TM with stay option can easily simulate an ordinary TM:**
  - It does not use the S option in any move.
- **An ordinary TM can easily simulate a TM with stay option.**
  - For each transition with the S option, introduce a new state, and two transitions
    - One transition moves the head right, and transits to the new state.
    - The next transition moves the head back to left for every possible tape symbol, and transits to the previous state.
- **Ordinary TMs and TMs with stay option have same power, and both of them accept Turing-recognizable languages.**

Multitape Turing Machines

- A **multitape Turing machine** is like an ordinary Turing machine with several tapes.
  - Each tape has its own head for reading and writing.
  - There are k tapes
  - Each tape has its own independent read/write head.
  - Initially the input appears on tape 1, and the others start out blank.
- The only fundamental difference from the ordinary TM is the state transition function.

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

- The entry $(q_i,a_1,\ldots,a_k) = (q_j,b_1,\ldots,b_k,D_1,\ldots,D_k)$ reads as :
  - If the TM is in state q_i and the heads are reading symbols a_1 through a_k ,
  - Then
    - The machine goes to state q_j ,
    - The heads write symbols b_1 through b_k , and
    - The heads move in the specified directions D_1 through D_k.

Multitape Turing Machines

- Multitape Turing machines appear to be more powerful than ordinary Turing machines, but we can show that they are equivalent in power.

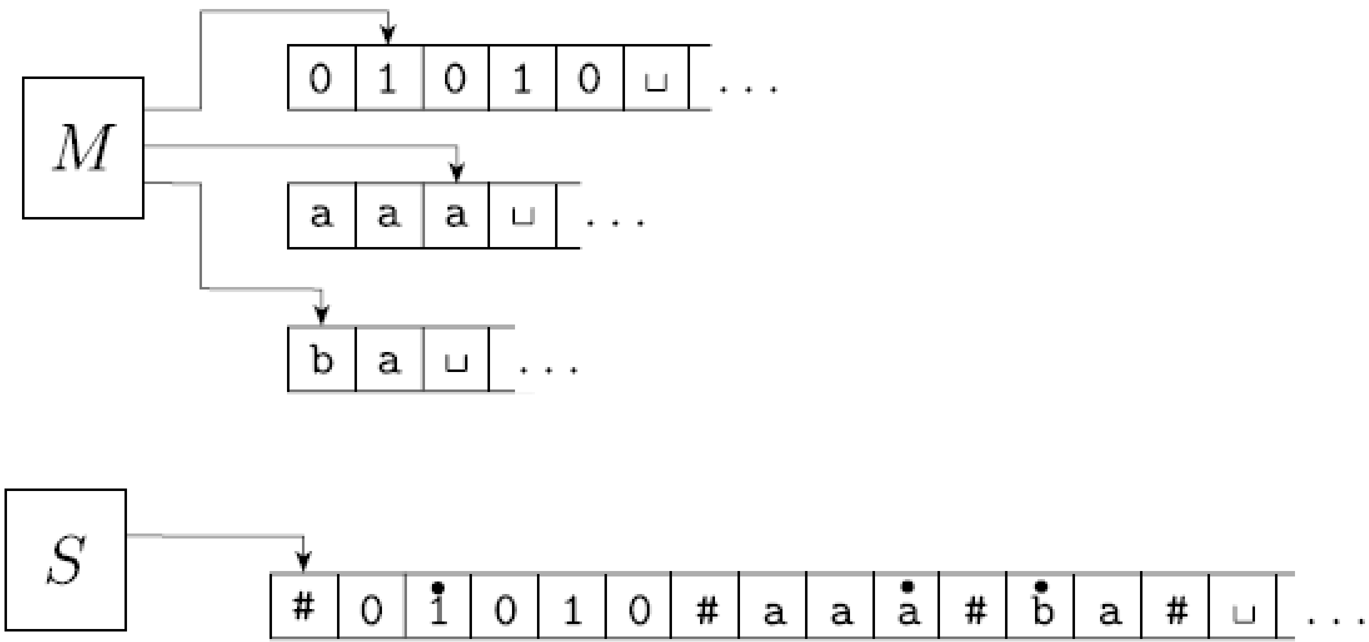
THEOREM:

- **Every multitape Turing machine has an equivalent single-tape Turing machine.**

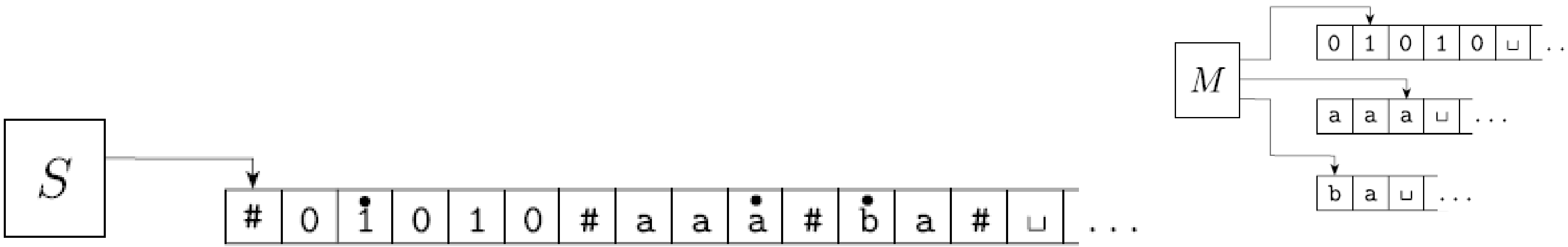
PROOF:

- **We show how to convert a multitape TM M to an equivalent single-tape TM S.**
- **The key idea is to show how to simulate M with S.**

Simulating Multitape TM with Ordinary TM



Simulating Multitape TM with Ordinary TM



- We use # as a delimiter to separate out the different tape contents.
- To keep track of the location of heads, we use additional symbols
  - Each symbol in tape alphabet Γ has a **“dotted”** version.
  - A dotted symbol indicates that the head is on that symbol.
  - Between any two #’s there is only one symbol that is dotted.
- Thus, we have one real tape with k “virtual’ tapes, and one real read/write head with k “virtual” heads.

Simulating Multitape TM with Ordinary TM

For a given input $w=w_1,\ldots,w_n$

- First S puts its tape into the format that represents all k tapes of M.

$$\#\dot{w}_1 w_2 \;\cdots\; w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \;\cdots\; \#$$

- To simulate a single move of M, S starts at the leftmost # and scans the tape to the rightmost #.
  - It determines the symbols under the “virtual” heads.
  - This is remembered in the finite state control of S.
- S makes a second pass to update the tapes according to M.
- If one of the virtual heads moves right to a #,
  - the rest of tape to the right is shifted to “open up” space for that “virtual tape”.
- If one of the virtual heads moves left to a #, it just moves right again.

Simulating Multitape TM With Ordinary TM

THEOREM: A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

PROOF:

- **A Turing-recognizable language is recognized by an ordinary (single tape)Turing machine.**
- **Every single tape TM is a special case of a multitape Turing machine.**
- **We showed that every multitape TM can be simulated by a single tape machine.**

- Thus, whenever needed or convenient, we can use multiple tape TMs.
- We can assume that these multitape TMS can always be converted to a single tape standard TM.

Nondeterministic Turing Machines

- **An ordinary TM is a deterministic machine.**
- The transition function of an **ordinary TM** is:

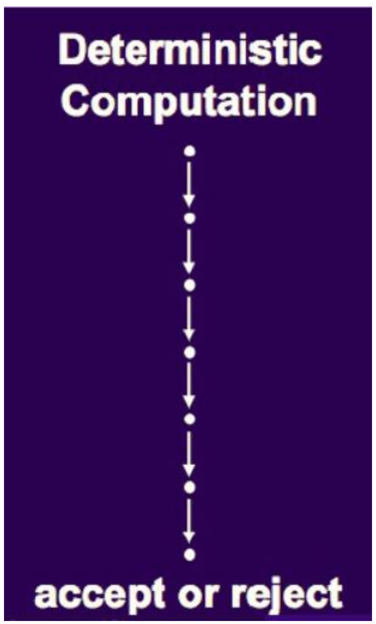
$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$$

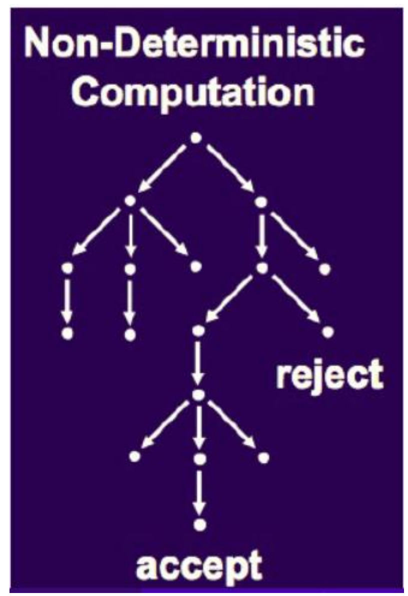
- A **nondeterministic TM** will proceed computation with multiple next configurations.
- The **transition function for a nondeterministic Turing Machine** has the form:

$$\delta: Q \times \Gamma \rightarrow \text{PowerSet}(Q \times \Gamma \times \{L,R\})$$

Nondeterministic Turing Machines

- A computation of a nondeterministic TM is a tree, where each branch of the tree is looks like a computation of an ordinary TM.
- If a single branch reaches the accepting state, the nondeterministic TM accepts, even if other branches reach the rejecting state.





Nondeterministic Turing Machines

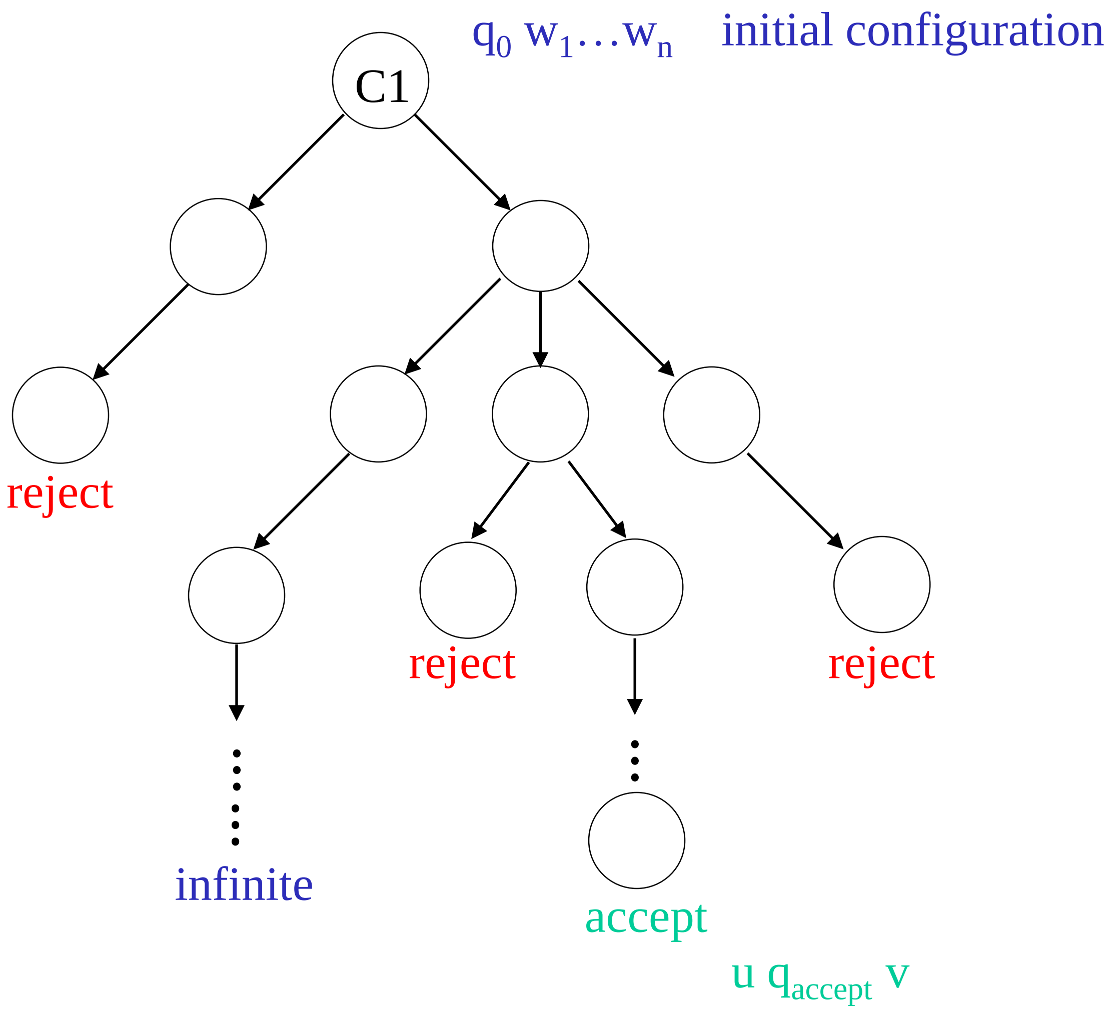
- What is the power of Nondeterministic TMs?
  - Is there a language that a nondeterministic TM can accept but no deterministic TM can accept? ➔ **NO**

THEOREM: Every nondeterministic Turing machine has an equivalent deterministic Turing Machine.

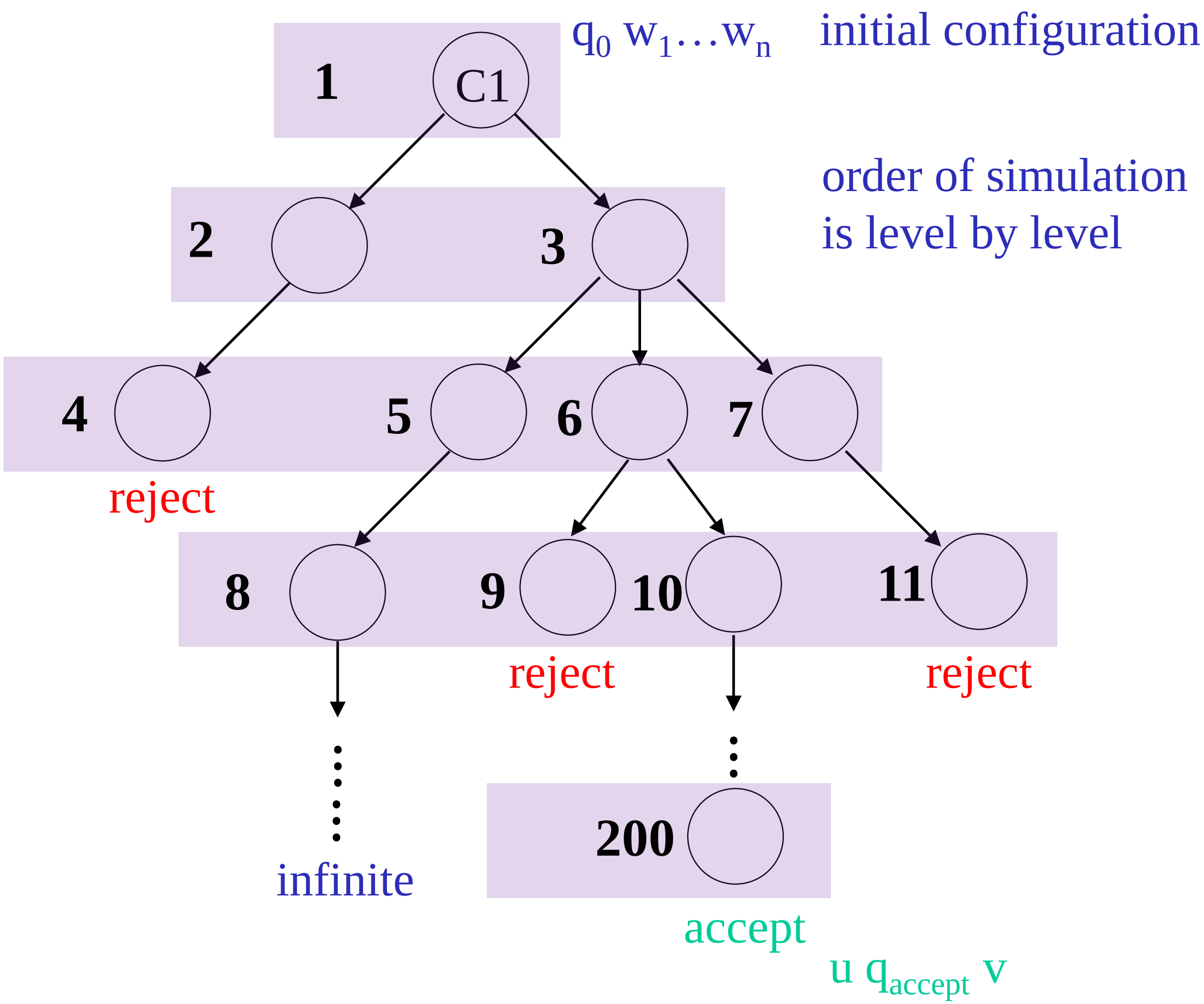
PROOF IDEA:

- **We can simulate any nondeterministic TM N with a deterministic TM D.**
- **The idea behind the simulation is to have D try all possible branches of N's nondeterministic computation.**
- **If D ever finds the accept state on one of these branches, D accepts.**
- **Otherwise, D's simulation will not terminate.**

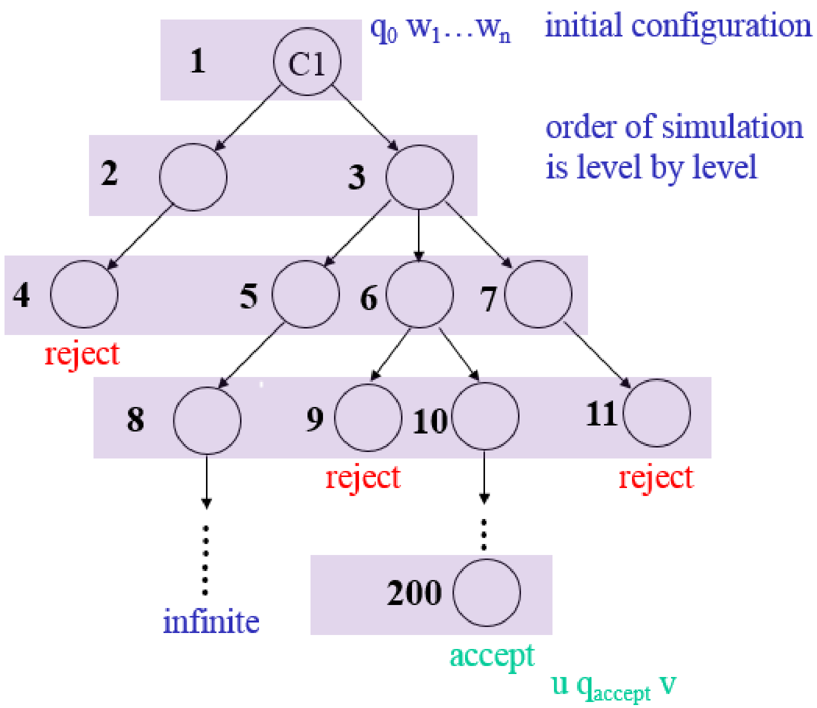
Nondeterministic Computation



Simulating Nondeterministic Computation



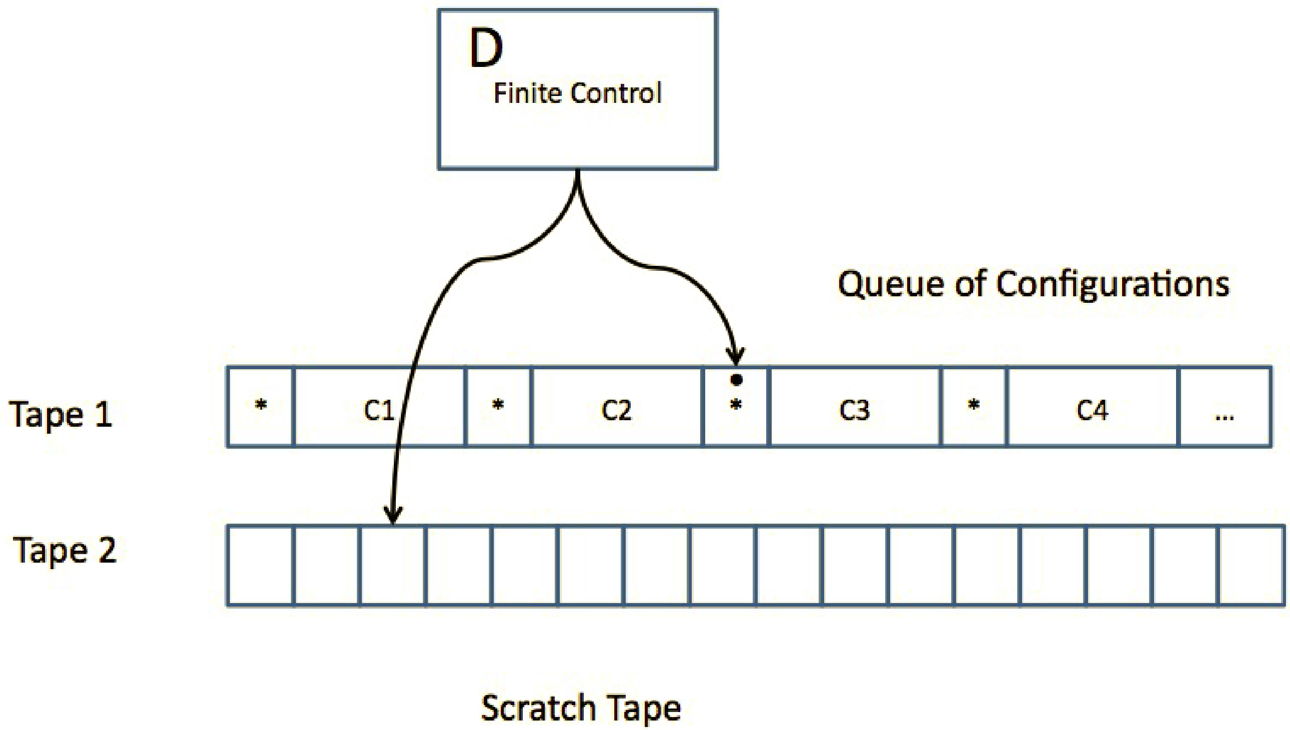
Simulating Nondeterministic Computation



- During simulation, D processes the configurations of N in a breadth-first fashion.
- Thus, D needs to maintain a queue of N's configurations
- D gets the next configuration from the head of the queue.
- D creates copies of this configuration (as many as needed)
- On each copy, D simulates one of the nondeterministic moves of N.
- D places the resulting configurations to the back of the queue.

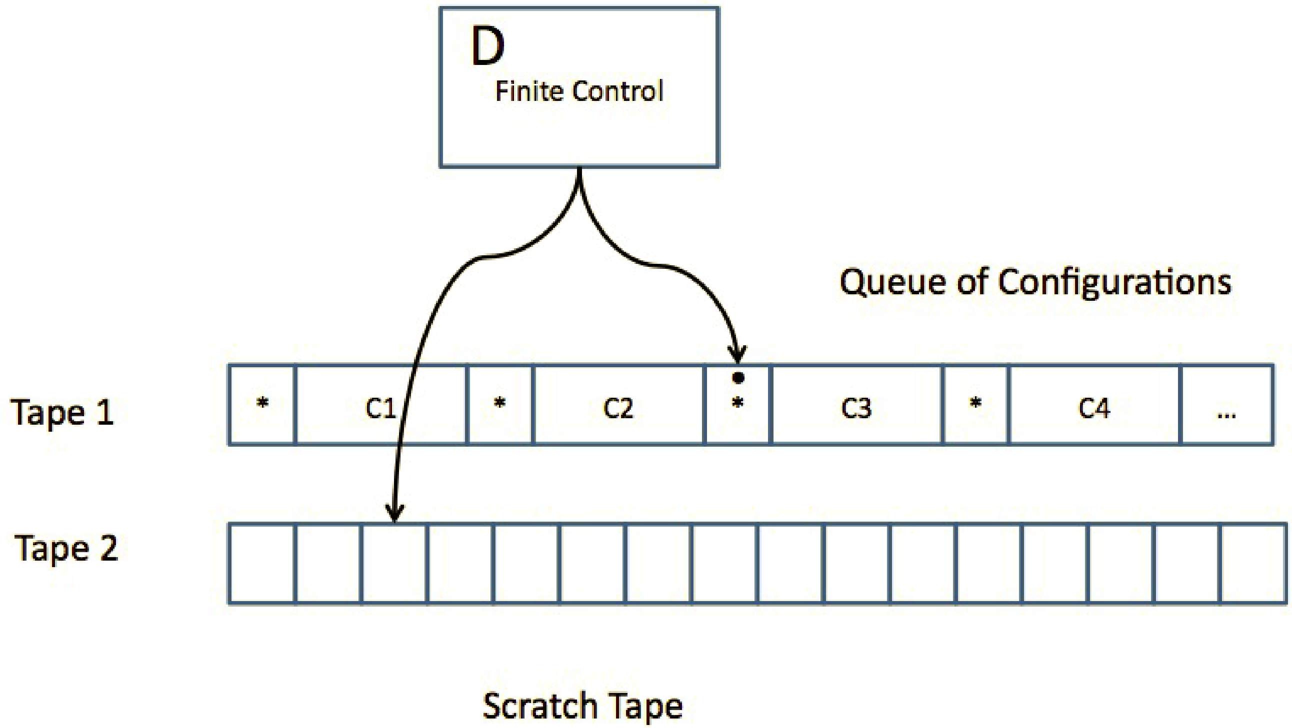
Structure Of Simulating Deterministic TM

- Nondeterministic TM N is simulated with 2-tape Deterministic TM D

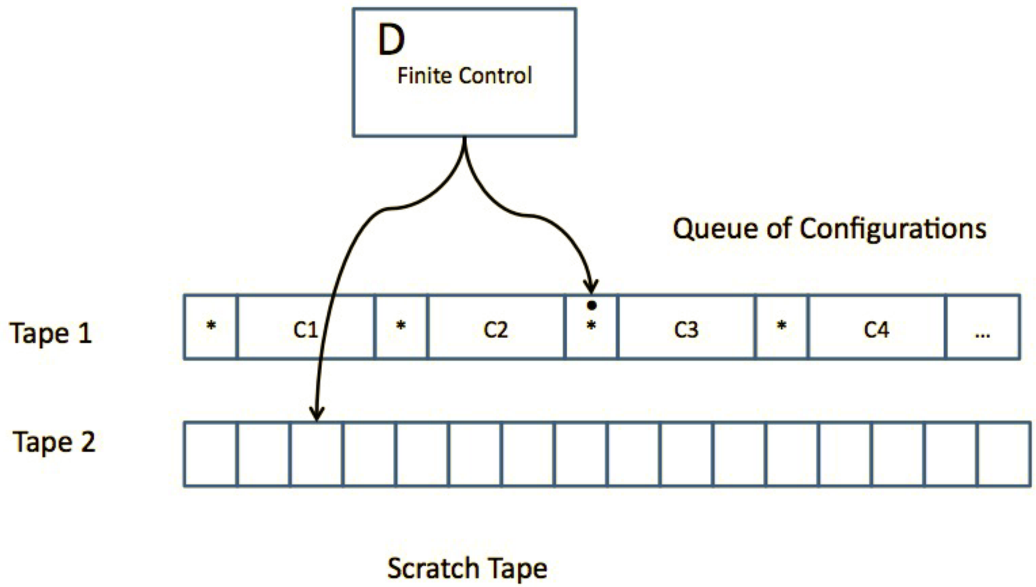


How D Simulates N

- Built into the finite control of D is the knowledge of what choices of moves N has for each state and input.

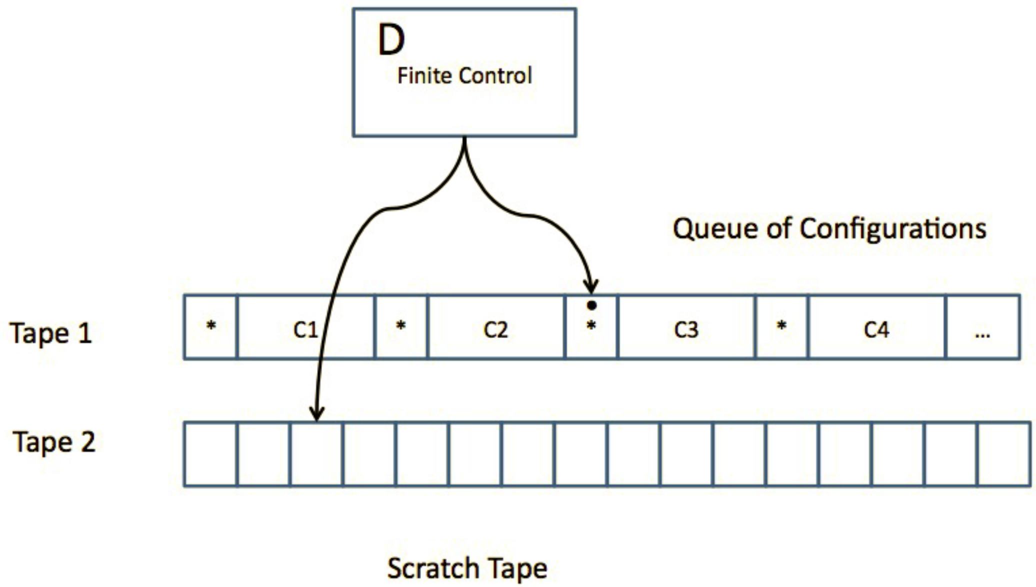


How D Simulates N



1. D examines the state and the input symbol of the current configuration (right after the dotted separator)
2. If the state of the current configuration is the accept state of N, then D accepts the input and stops simulating N.
3. D copies k copies of the current configuration to the scratch tape.
4. D then applies one nondeterministic move of N to each copy.

How D Simulates N



5. D then copies the new configurations from the scratch tape, back to the end of tape 1 (so they go to the back of the queue), and then clears the scratch tape.
6. D then returns to the marked current configuration, and “erases” the mark, and “marks” the next configuration.
7. D returns to step 1, if there is a next configuration. Otherwise rejects.

Nondeterministic Turing Machines

COROLLARY:

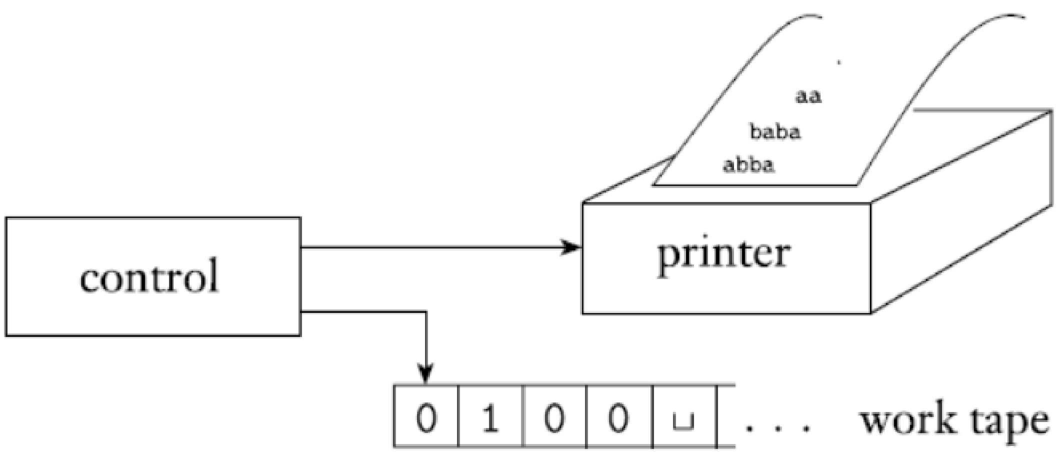
- **A language is Turing-recognizable if and only if some nondeterministic TM recognizes it.**

COROLLARY:

- **A language is decidable if and only of some nondeterministic TM decides it.**

Enumerators

- As we mentioned earlier, we can use the term **recursively enumerable language** for **Turing-recognizable language**.
- That term originates from a type of Turing machine variant called an **enumerator**.
- Loosely defined, an **enumerator** is a **Turing machine with an attached printer**.
  - The Turing machine can use that printer as an output device to print strings.
  - Every time Turing machine wants to add a string to the list, it sends the string to printer.



Enumerators

- The enumerator E starts with a blank input tape.
- If it does not halt, it may print an infinite list of strings.
- The strings can be enumerated in any order; repetitions are possible.
- The language of the enumerator is the collection of strings it eventually prints out.

Enumerators

THEOREM: A language is Turing-recognizable if and only if some enumerator enumerates it.

PROOF:

If-part: If an enumerator E enumerates the language A then a TM M recognizes A.

M = "On input w

1. Run E. Every time E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept."

- Clearly M accepts only those strings that appear on E's list.

Enumerators

THEOREM: **A language is Turing-recognizable if and only if some enumerator enumerates it.**

PROOF:

Only-If-part: If a TM M recognizes a language A, we can construct the following enumerator for A. Assume s_1, s_2, … is a list of possible strings in Σ^*.

E = "Ignore the input

1. Repeat the following for i = 1, 2, …
2. Run M for i steps on each input s_1, s_2, … s_i.
3. If any computations accept, print out corresponding s_k ."

If M accepts a particular string, it will appear on the list generated by E (in fact infinitely many times)

Church-Turing Thesis

- An **algorithm** is a finite sequence of precise instructions for performing a computation or for solving a problem.“
- In early 20th century, there was no formal definition of an algorithm.
- In 1936, Alonzo Church and Alan Turing came up with formalisms to define algorithms. These were shown to be equivalent, leading to the

Church-Turing thesis

Each algorithm can be implemented by some Turing machine.