CS 600
Evgenios Kornaropoulos

Lecture 3

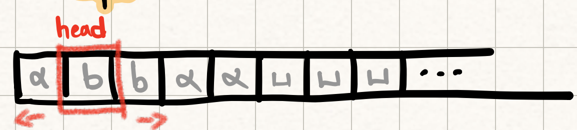
Turing Machines (Sipser, Chapter 3.1)

- So far we have seen two computational models, "Finite Automata" and "Pushdown Automata". Finite automata have no access to memory and Pushdown automata have unlimited memory (in a form of a stack) but usable only in the last in first out manner. They both have limited capabilities which is why we switch to a more powerful model.

- Turing machines (TM) were proposed by Alan Turing in 1936 and they can do everything a real computer can do. Unfortunately, there are certain problems that even TMs cannot solve ⇒ problems beyond the theoretical limits of computation.

- The TM model uses a tape that has a leftmost end but not a rightmost end.

Tape:



More importantly, we can now read and write on any location of the tape. To capture this functionality, we have to introduce notation that specifies if the head of the tape is moving to the left or right.

- Initially, the tape contains only the input string starting from the leftmost location and the blank character "⊔" everywhere else.

- The outputs "accept" and "reject" are obtained by entering the designated accepting and rejecting states. These states are trap states (recall that in Finite and Pushdown automata they are not trap states), and they terminate the execution immediately.

Formal Definition of a Turing Machine

The transition function δ of a Turing machine takes the form:

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

- Q (left): the state that the machine is currently at
- Γ (left): the symbol at the current position of the machine head
- Q (right): the next state that the machine is moving to
- Γ (right): the new symbol that is written at the current position of the machine head
- $\{L, R\}$: the direction that the head moves after writing

Definition 3.3: 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 all states
2. Σ is the input alphabet not containing the blank symbol "$\sqcup$"
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{accept} \in Q$ is the accept state
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$

- Notice that Σ does not contain "$\sqcup$", therefore the first "$\sqcup$" appearing on the tape marks the end of the input.

- If the machine ever tries to move to the left of the leftmost cell of the tape, then the head stays in the same place for that move.

- As the TM computes, changes happen in:
  1. the current state
  2. the current tape contents
  3. the current head location

An instantiation/setting of the above three parameters is called a configuration of the Turing machine.

- A compact representation of configurations: The configuration "uqv" is made up of two strings u and v over the tape alphabet as well as a state q.

State: q_7

Tape:	1	0	1	1	$\boxed{0}$	1	1	1	1	$\sqcup$	$\sqcup$	...
					head							

Representation

$$\rightsquigarrow \underbrace{1011}_{u}\underbrace{q_7}_{q}\underbrace{01111}_{v}$$

- The tape contains uv, i.e., concatenation of u and v.
- State q "cuts" the tape in two pieces, its position indicates that the next symbol that the head reads is the first symbol of v.

- Configuration C_1 yields configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single transition.

More formally: Suppose symbols $a, b, c \in \Gamma$ and strings $u, v \in \Gamma^*$ and $q_i, q_j \in Q$.

e.g., abccbabb....

We say that: $u a q_i b v$ yields $u q_j a c v$
if the transition $\delta(q_i, b) = (q_j, c, L)$ is part of δ definition.

We say that: $u\alpha q_i b v$ yields $u\alpha c q_j v$
if the transition $\delta(q_i, b) = (q_j, c, R)$ is part of δ definition.

- A Turing machine **accepts** input w if a sequence of configurations $C_1, C_2, \dots, C_k$ exists, where:
  1. C_1 is the start state configuration of M on input w, i.e., $C_1 = q_0 w$.
  2. Each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration

- The collection of strings that M accepts is the language **recognized** by M, denoted by L(M).

Definition 3.5: Call a language Turing-recognizable if some Turing machine recognizes it.

⚠ A TM operating on an input w can have three outcomes, accept, reject, or loop. Loop means that the machine does not halt. Distinguishing a machine that is looping from one that is taking too long is difficult.

- The TMs that halt on all inputs, i.e., they always make a decision to accept or reject, are called **deciders**.

Deciding vs. Recognizing:

- Turing machine M **decides** L if and only if
  1. M outputs "accept" for every $w \in L$
  2. M outputs "reject" for every $w \notin L$

- Turing machine M **recognizes** L if and only if M outputs "accept" for all and only the input strings $w \in L$ → this definition allows M to loop if $w \notin L$.

- If M decides L, then M recognizes L (the other direction is not true).

• Example 3.7: A TM that decides the language consisting of all strings of 0s whose length is a power of two, i.e., $L = \{ 0^{2^n} \mid n \geq 0 \}$

Let's think about it: There are some easy corner cases that we can handle fast.
- For example, if the input is the empty string or the input is a symbol from $\Gamma \setminus \Sigma$ → Reject.
  ↑ A symbol in the tape alphabet Γ but not input alphabet Σ
- If we have an $\overbrace{\text{odd}}^{>1}$ number of 0s, then the input cannot be in L → Reject
- We need to check if the the number of 0s is a power of 2, e.g. $w = 0000 = 0^{2^2}$
  Suppose you want to generate 2^n zeros, then most probably you would:

String: 0 $\xrightarrow{double}$ 00 $\xrightarrow{double}$ 0000 $\xrightarrow{double}$ 00000000 $\xrightarrow{double}$ ... $\xrightarrow{double}$ 00.....00

\# of 0s: 2^0 2^1 2^2 2^3 2^n

- What if instead of doubling to generate a new string, we instead halve the number of 0s.

Input: 00000000

00000000 $\xrightarrow{halve}$ ~~0000~~0000 $\xrightarrow{halve}$ ~~000000~~00 $\xrightarrow{halve}$ ~~0000000~~0

2^3 2^2 2^1 2^0

If we count how many times we halved, we can calculate the "n" in 0^{2^n}

More importantly, if by halving we reach a point where we have more 0s but we cannot halve anymore, then the input is not a member of L.

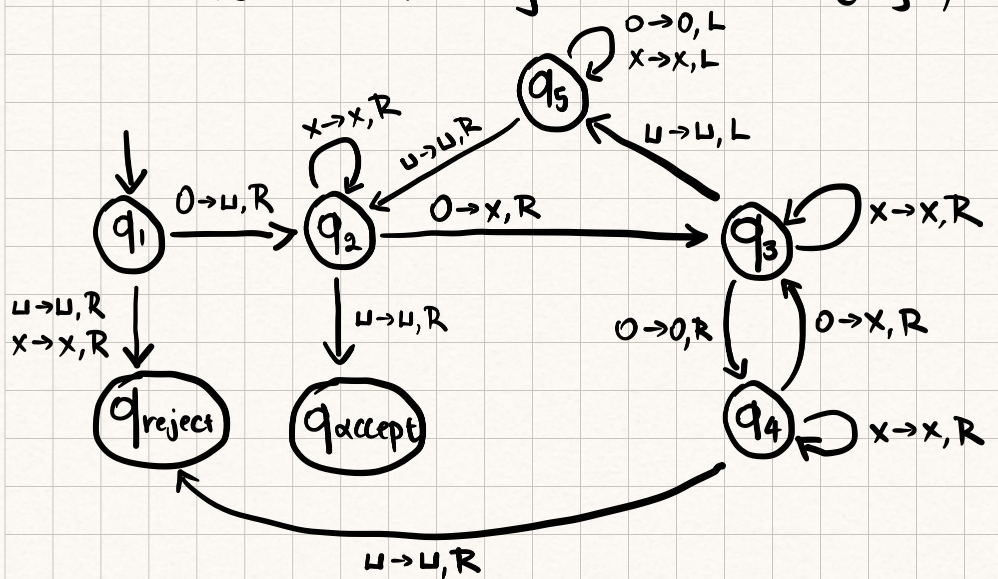
-Notice that when we halved, we crossed out a sequence starting from the leftmost remaining 0. To do that, we need to know the total number of remaining 0s. To perform this in a TM we need to do <u>one pass</u> to count the # of 0s and <u>another pass</u> to cross out the right number of them.

- There is a smarter way that uses <u>only one pass</u> → Cross out every other 0.

$\xrightarrow{halve}$ ~~0~~0~~0~~0~~0~~0~~0~~0 $\xrightarrow{halve}$ ~~000~~0~~000~~0 $\xrightarrow{halve}$ ~~0000000~~0

- One last thing: Since we need to do a lot of back and forth on the tape, we need to mark the beginning of the tape → Cross out the first 0 using ⊔

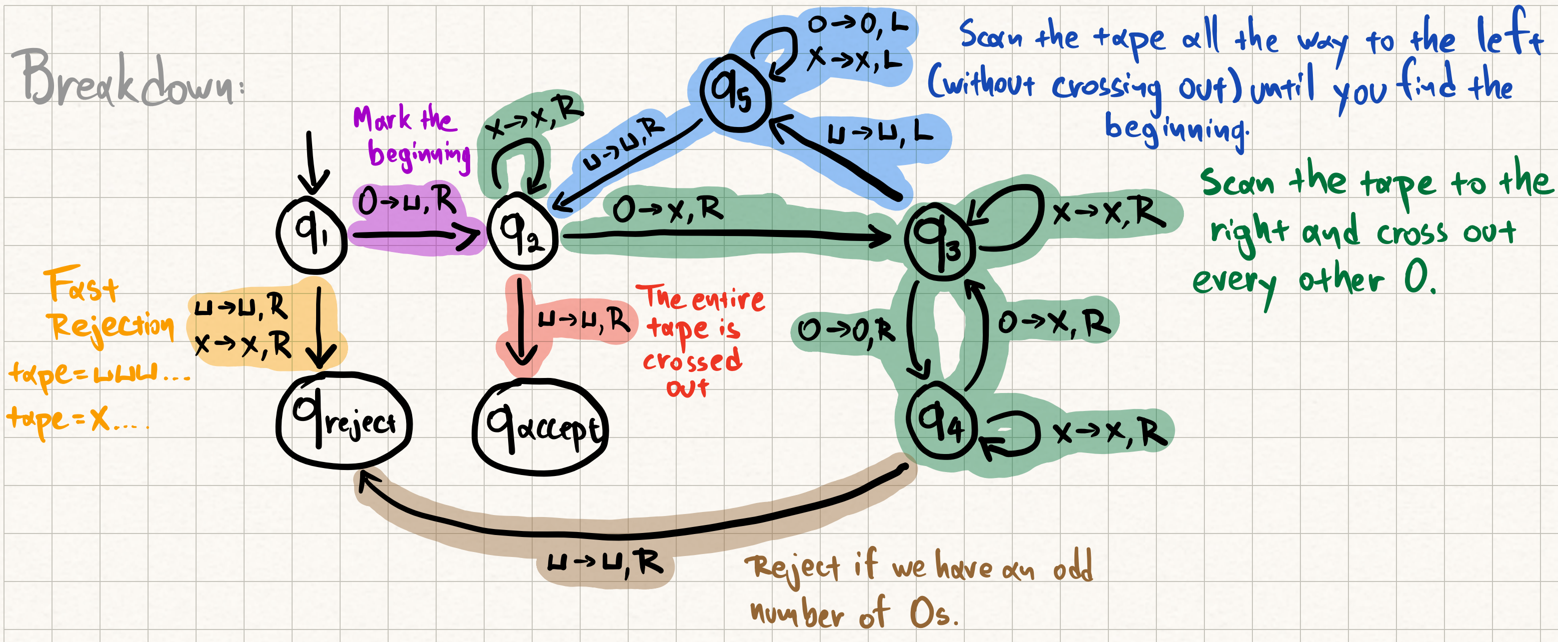
- We define the following TM with $\Sigma = \{0\}$, $\Gamma = \{⊔, 0, x\}$



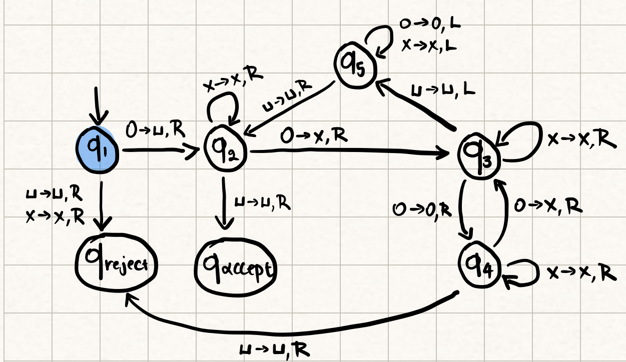
Transition Function:

	0	x	⊔
q_1	q_2,⊔,R	q_{rej},⊔,R	q_{rej},x,R
q_2	q_3,x,R	q_2,x,R	q_{acc},⊔,R
q_3	q_4,0,R	q_3,x,R	q_5,⊔,L
q_4	q_3,x,R	q_4,x,R	q_{rej},⊔,R
q_5	q_5,0,L	q_5,x,L	q_2,⊔,R
q_{accept}	⊥	⊥	⊥
q_{reject}	⊥	⊥	⊥

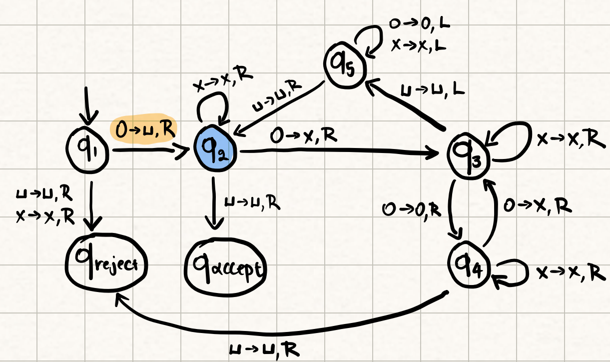
Breakdown:



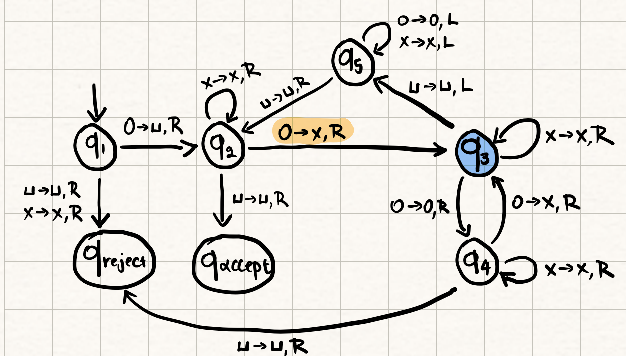
Example Run w=0000



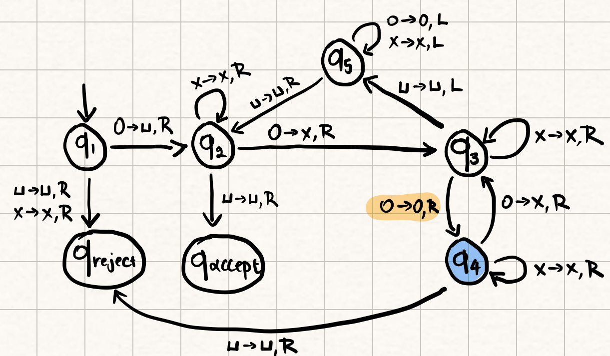
Configuration: $q_1 0000$ Tape: 0000



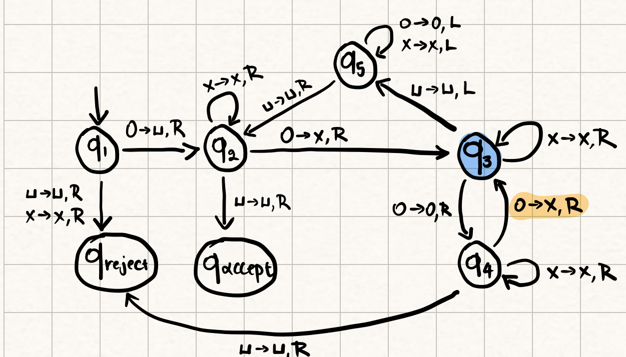
Configuration: $\sqcup q_2 000$ Tape: ⊔000



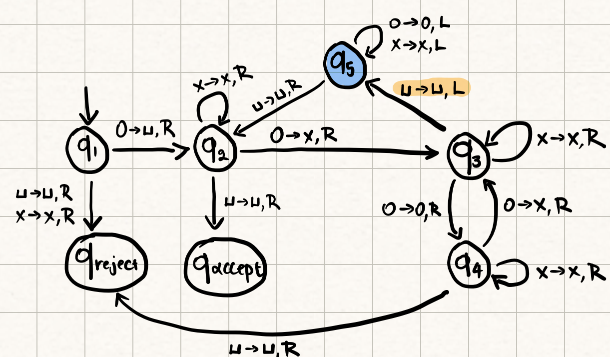
Configuration: $\sqcup x q_3 00$ Tape: ⊔x00



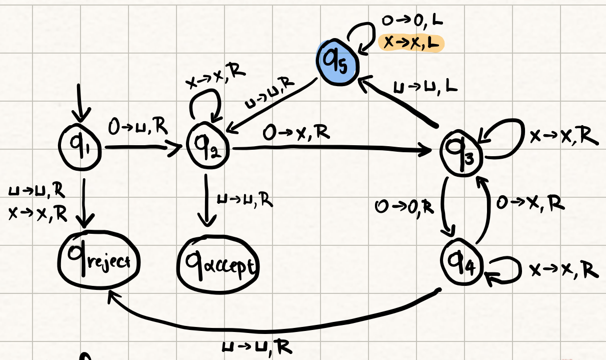
Configuration: $\sqcup x 0 q_4 0$ Tape: ⊔x00



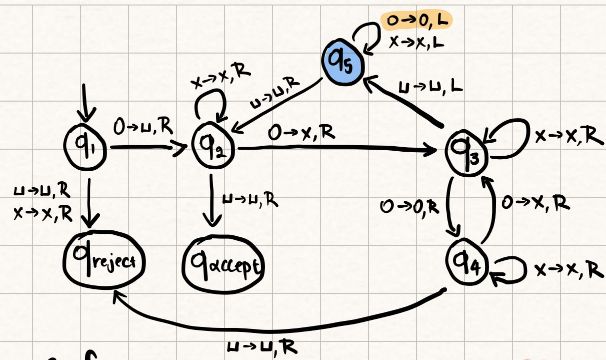
Configuration: $\sqcup x 0 x q_3$ Tape: ⊔x0x⊔



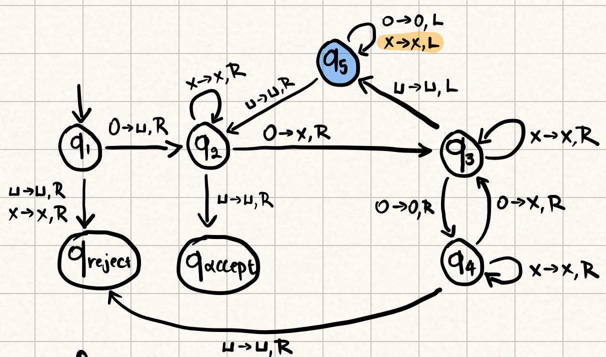
Configuration: $\sqcup x 0 q_5 x$ Tape: ⊔x0x



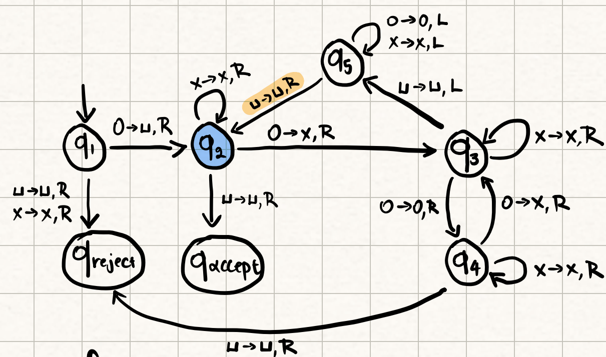
Configuration: $\sqcup x q_5 0 x$ Tape: $\sqcup x \boxed{0} x$



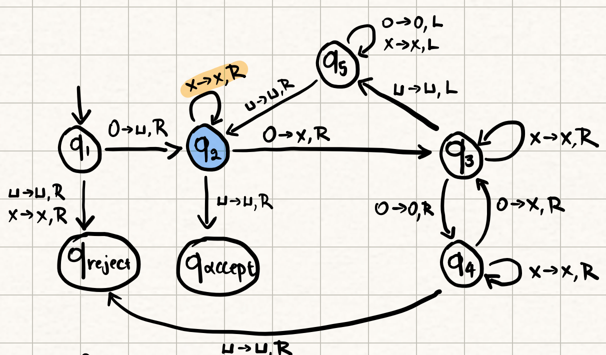
Configuration: $\sqcup q_5 x 0 x$ Tape: $\sqcup \boxed{x} 0 x$



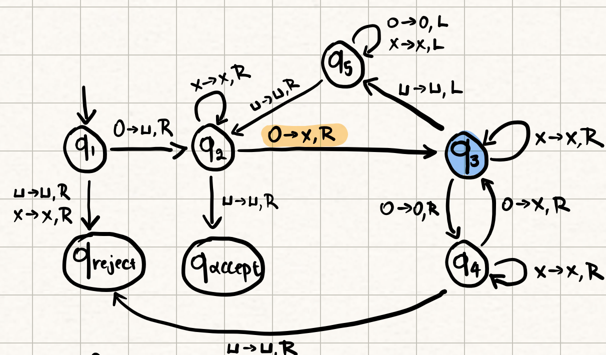
Configuration: $q_5 \sqcup x 0 x$ Tape: $\boxed{\sqcup} x 0 x$



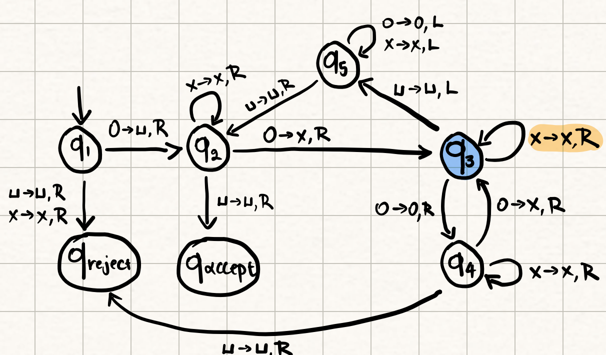
Configuration: $\sqcup q_2 x 0 x$ Tape: $\sqcup \boxed{x} 0 x$



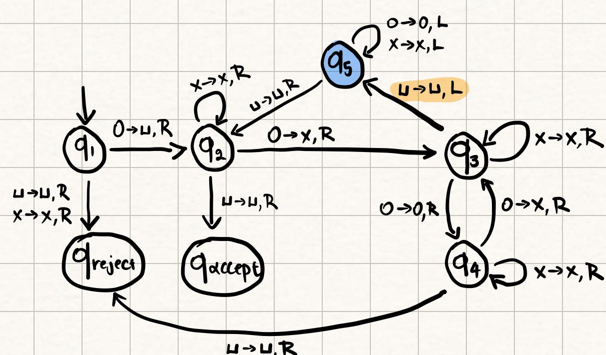
Configuration: $\sqcup x q_2 0 x$ Tape: $\sqcup x \boxed{0} x$



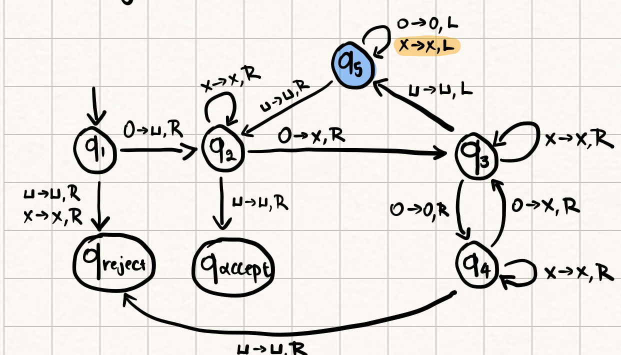
Configuration: $\sqcup x x q_3 x$ Tape: $\sqcup x x \boxed{x}$



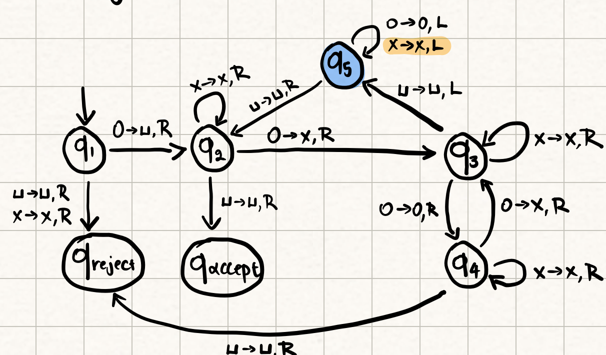
Configuration: $\sqcup x x x q_3$ Tape: $\sqcup x x x \boxed{\sqcup}$



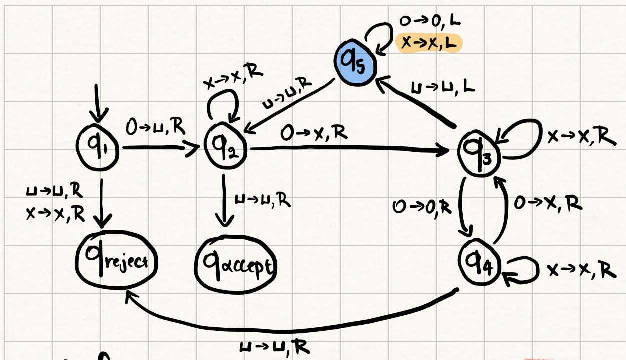
Configuration: $\sqcup x x q_5 x$ Tape: $\sqcup x x \boxed{x}$



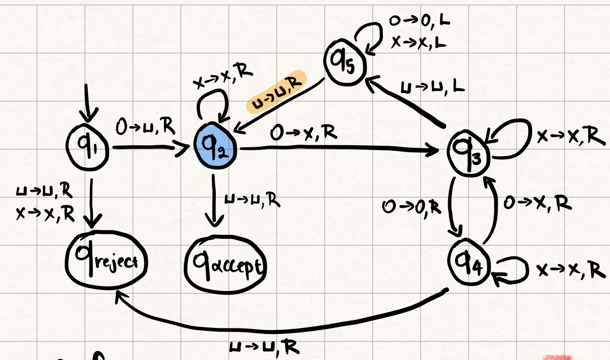
Configuration: $\sqcup x q_5 x x$ Tape: $\sqcup x \boxed{x} x$



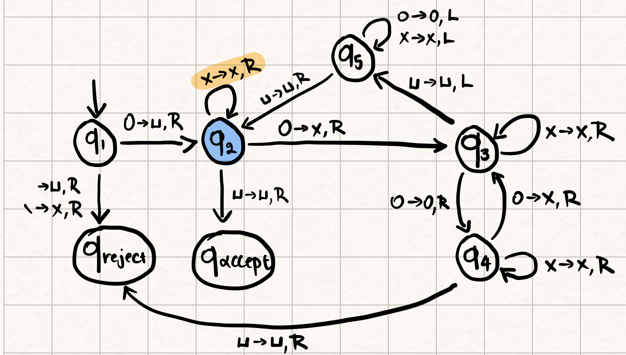
Configuration: $\sqcup q_5 x x x$ Tape: $\sqcup \boxed{x} x x$



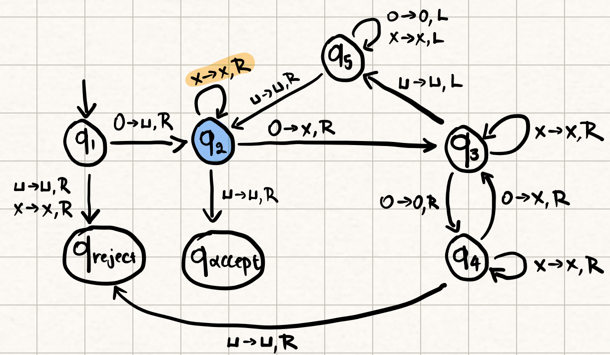
Configuration: q_5⊔XXX Tape: ⊔XXX



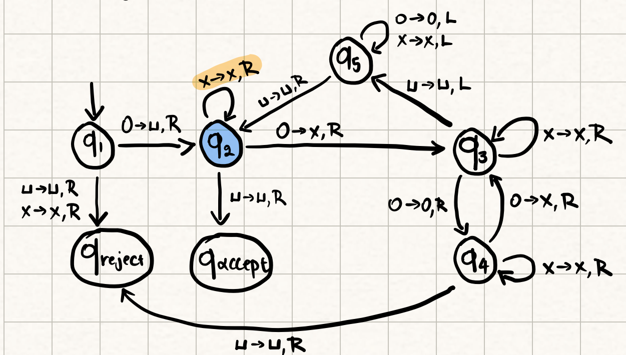
Configuration: ⊔q_2XXX Tape: ⊔XXX



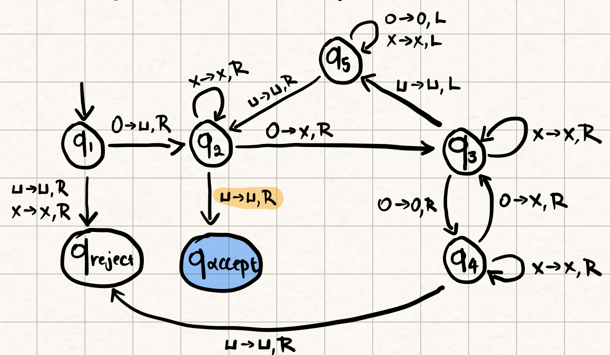
Configuration: ⊔Xq_2XX Tape: ⊔XXX



Configuration: ⊔XXq_2X Tape: ⊔XXX



Configuration: ⊔XXXq_2 Tape: ⊔XXX⊔



Configuration: ⊔XXX⊔q_{accept} Tape: ⊔XXX⊔⊔

Quiz 3.1: Suppose we compute on the above TM. Which of the following configurations cannot be seen in this TM?

A) $q_1$00000000
B) ⊔x0$q_4$0
C) ⊔XXXXX⊔q_{accept}
D) ⊔XXXXXXX⊔q_{accept}

-8-

2. Effective calculability. Abbreviation of treatment.

A function is said to be 'effectively calculable' if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea it is nevertheless desirable to have some more definite, mathematically expressible definition. Such a definition was first given by Gödel at Princeton in 1934 (Gödel [2], 26) following in part an unpublished suggestion of Herbrand, and has since been developed by Kleene (Kleene [2]). We shall not be concerned much here with this particular definition. Another definition of effective calculability has been given by Church (Church [3], 356-358) who identifies it with λ-definability. The author has recently suggested a definition corresponding more closely to the intuitive idea (Turing [1], see also Post [1]). It was said above "a function is effectively calculable if its values can be found by some purely mechanical process." We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and an identification of computability[3] with effective calculability.

[3] We shall use the expression 'computable function' to mean a function calculable by a machine, and let 'effectively calculable' refer to the intuitive idea without particular identification with any one of these definitions. We do not restrict the values taken by a computable function to be natural numbers; we may for instance have computable propositional functions.

It is not difficult though somewhat laborious, to prove these

Ph.D. Thesis of Alan Turing titled "Systems of Logic Based on Ordinals", 1938

The Church-Turing Thesis (Sipser, Ch. 3.3)

- No computational procedure will be considered as an algorithm unless it can be represented as a Turing machine

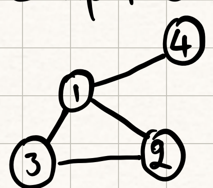
- Intuitive notion of algorithms $\Longleftrightarrow$ Turing machine algorithms

What is the right level of detail when describing TM algorithms?

1. **Formal Description:** Provide a detailed description of TM's states, transition function, etc.
2. **Implementation Description:** Use text to describe the way that it moves its head and the way it stores data on its tape.
3. **High-level Description:** Use text to describe the algorithm ignoring implementation details (head movement etc.)

- TMs are powerful. They can handle/solve problems beyond regular languages. They can handle languages that concern all kinds of mathematical objects.

For example:
Graph G



$$G \xrightarrow{\text{Encoding as a string}} \langle G\rangle = \underbrace{(1,2,3,4)}_{\text{List of nodes}}\underbrace{((1,2),(2,3),(3,1),(1,4))}_{\text{List of edges}}$$

Properly formed input strings:
- The list of nodes should contain no repetitions
- List of nodes: decimal numbers, List of edges: pairs of decimal numbers
- Every node on edge list should appear on node list

Thus, we can have a TM that decides language

$$L = \{\langle G\rangle \mid G \text{ is a connected undirected graph}\}$$

Decidability (Sipser, Chapters 4.1 and 4.2)

- Let's investigate the power of TM/algorithms to solve problems. We will see that some problems can be solved algorithmically but certain problems cannot.

⇓

Explore the limits of algorithmic solvability

· Decidable Languages

We give an algorithm for testing whether a finite automaton accepts a string

Theorem 4.1: Language $L = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ is a decidable language

Proof:

High-level description

M = "On input $\langle B, w\rangle$, where B is a 5-tuple describing a DFA and w is a string:
1. Simulate B on input w
2. If the simulation ends in B's accept state, then M accepts. If it ends in a non-accepting state, then M rejects."

First, M checks if the input string is a properly formed $\langle B, w\rangle$ encoding (i.e., a 5-tuple describing a DFA followed by w), if not reject.

M carries the simulation directly ⇒ Keeps track of B's current state and its position in the input w. by writing on M's tape.

Notice that B is a DFA therefore, it performs a single pass on input w, which means that it will terminate in $|w|$ steps. ∎

Undecidability

One of the most philosophically important findings:

"There are problems that are algorithmically unsolvable"

Goal: Learn techniques to prove that a problem is computationally unsolvable.

[Theorem 4.11: The language $L_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ is undecidable.

Some observations first:
- This theorem shows that recognizers are more powerful than deciders.
- Requiring a TM to halt on all inputs restricts the languages that it can process.

For example, the following simple TM recognizes L_{TM} → not "decides"

U = "On input $\langle M, w\rangle$, where M is a TM and w is a string:
1. Simulate M on input w.
2. If M ever enters its accept state, then U accepts, if M ever enters its reject state, then U rejects."

* It is possible that M loops on input w, which is why U recognizes L_{TM} but does not decide L_{TM}.

The above is an example of a universal Turing machine that is capable of simulating any other TM M given its description.
⇓
Predecessor of modern computer "one machine that runs arbitrary "machines" based on the program"

The proof of Thm 4.11 is based on the Diagonalization method discovered by Cantor in 1873. The motivating question was:

"If we have two infinite sets how can we tell if one is larger than the other or whether they are of the same size?"
↳ If we start counting to compare their relative sizes we will never finish.

[Key Observation: For the case of finite sets, two sets have the same size if the elements of one set can be paired with the elements of the other set ⇒ Extend this to infinite sets!
comparison without counting

Some definitions before introducing the diagonilization method.
Let A, B be two sets and f be a function from A to B.

- Function f is injective (one-to-one) if it never maps two different elements to the same place, i.e., $\forall a, b \in A,\ a \neq b \Rightarrow f(a) \neq f(b)$

- Function f is **surjective (onto)** if it hits every element of B
  i.e., $\forall b \in B$, $\exists a \in A$ such that $f(a) = b$

- If function f that is both one-to-one and onto is called a **correspondence**.

* We say that A and B have the **same size** if there is a correspondence between them.

<u>Example</u>: Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ be the set of natural numbers and
$\mathcal{E} = \{2, 4, 6, \ldots\}$ be the set of even natural numbers.
We can prove that these infinite sets have the same size by providing a correspondence from $\mathbb{N}$ to $\mathcal{E}$.

$f(n) = 2n$ visually $\rightsquigarrow$

n	$f(n)$
1	2
2	4
3	6
4	8
$\vdots$	$\vdots$

* Counterintuitive example since $\mathcal{E}$ is a proper subset of $\mathbb{N}$, $\mathcal{E} \subset \mathbb{N}$.

- A set S is **countable** if ① either it is finite, or ② it has the same size as $\mathbb{N}$.

<u>Another Example</u>: The set of positive rational numbers $\mathbb{Q} = \left\{ \frac{m}{n} \middle| m, n \in \mathbb{N} \right\}$ has the same size as $\mathbb{N}$.

*Even more counterintuitive!

To prove that they have the same size, we give a correspondence.
Create an infinite matrix to list all members of $\mathbb{Q}$

$$
\begin{array}{c|ccccc}
 & \multicolumn{5}{l}{\text{Increase Denominator} \longrightarrow} \\
\hline
\text{Increase} & \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\
\text{Numerator} & \frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} \\
\downarrow & \frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} & \frac{3}{5} & \cdots \\
 & \frac{4}{1} & \frac{4}{2} & \frac{4}{3} & \frac{4}{4} & \frac{4}{5} \\
 & \frac{5}{1} & \frac{5}{2} & \frac{5}{3} & \frac{5}{4} & \frac{5}{5} \\
 & & \vdots
\end{array}
$$

* The proposed function must not have the same output for two different inputs. Notice that some numbers are repeated in this infinite matrix $1 = \frac{1}{1} = \frac{2}{2} = \frac{3}{3} = \frac{4}{4} = \frac{5}{5}$ and $2 = \frac{2}{1} = \frac{4}{2}$

* Also we must output every member of set $\mathbb{Q}$.

Thus, this f: $\mathbb{N} = (1, 2, 3, 4, \ldots)$ is not a correspondence because it will never reach the second row

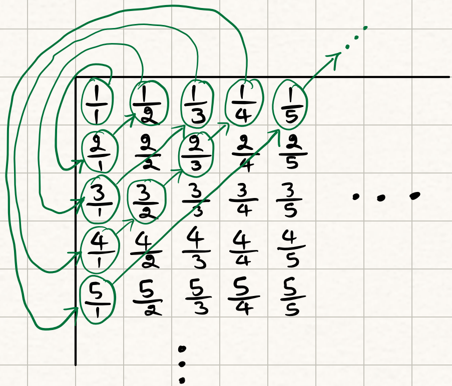
$$
\begin{array}{|ccccc}
\hline
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} \\
\frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} & \frac{3}{5} & \cdots \\
\frac{4}{1} & \frac{4}{2} & \frac{4}{3} & \frac{4}{4} & \frac{4}{5} \\
\frac{5}{1} & \frac{5}{2} & \frac{5}{3} & \frac{5}{4} & \frac{5}{5} \\
 & & \vdots
\end{array}
$$

(*) List the members on the diagonals

$$
\begin{array}{|ccccc}
\hline
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} \\
\frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} & \frac{3}{5} & \cdots \\
\frac{4}{1} & \frac{4}{2} & \frac{4}{3} & \frac{4}{4} & \frac{4}{5} \\
\frac{5}{1} & \frac{5}{2} & \frac{5}{3} & \frac{5}{4} & \frac{5}{5} \\
 & & \vdots
\end{array}
$$

<u>But</u> if we simply list them like that, we create repetitions:
$\left(\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{2}{2}, \frac{1}{3}, \ldots\right)$

× not a correspondence



Skip repeated entries to ensure one-to-one property

⇓

Correspondence between $\mathbb{N}$ and $\mathbb{Q}$.

- For some infinite sets there exists no correspondence with $\mathbb{N}$. Such a set is called **uncountable**. (⇒ Some infinite sets are larger than other infinite sets)

Example: The set of real numbers $\mathbb{R}$ is uncountable.

Proof: We proceed with a proof by contradiction. Suppose for the sake of contradiction that exists a correspondence f between $\mathbb{N}$ and $\mathbb{R}$. We will provide a number $r \in \mathbb{R}$ that does not appear as an output of f.

An illustration:

n	$f(n)$
1	3.14159...
2	55.55...
3	0.1234...
4	0.5000
⋮	

← Hypothetical correspondence

define r such that, the i-th decimal of r is different from the i-th decimal of $f(i)$.

$r = 0.2415\ldots$

$f(1)$: $\neq 1$, $f(2)$: $\neq 5$, $f(3)$: $\neq 3$, $f(4)$: $\neq 0$

If we construct an r such that the i-th decimal is different from the i-th decimal of $f(i)$, then we know that r is not equal to $f(n)$ for any value of n. ∎

Before we see the proof for Theorem 4.11, let's first prove that there are languages that are not Turing-recognizable.

~~Corollary~~ Theorem 4.18: Some languages are not Turing-recognizable.

Lemma-A: For any alphabet Σ, **the set of strings Σ^* is countable**

Proof for Lemma-A: Make a list that covers all the members of the infinite set Σ^*.

Let $\Sigma = \{a, b, c\}$

List LS_{Σ^*}: $\underbrace{a, b, c}_{\text{all strings with characters from } \Sigma \text{ with length 1.}}$, $\underbrace{aa, ab, ac, ba, bb, bc, ca, cb, cc}_{\text{all strings with characters from } \Sigma \text{ with length 2}}, \ldots$

It is easy to see that the index of its entry of LS_{Σ^*} can be used as a $\mathbb{N}$ value towards building a correspondence with LS_{Σ^*}. Thus, the set of strings Σ^* is countable. ∎

Corollary-A: We know that every Turing machine M can be encoded as a finite string $\langle M \rangle$. If we omit from Σ^* the strings that do not encode a TM, then what is left is a subset of Σ^* which we know is countable. Thus, the set of all TM is countable.

An infinite binary sequence is an unending sequence of 0s and 1s.

Lemma-B: The set of infinite binary sequences $\mathbb{B}$ is uncountable.

Proof for Lemma-B: By diagonilization. Suppose for the sake of contradiction that $\mathbb{B}$ is countable. Then, there exists a correspondence with $\mathbb{N}$. We can create an infinite binary sequence r that does not appear as an output of $f \Rightarrow f$ is not onto $\Rightarrow f$ is not a correspondence. Iterate through the list implied by f, for the i-th entry of the list, check the i-th bit of $f(i)$ and assign the opposite to the i-th bit of r.

$f(1) = 10110\ldots \quad \neq \quad r = 0\_\_\_\ldots$

$f(2) = 10011\ldots \quad \neq \quad r = 01\_\_\ldots$

$f(3) = 01101\ldots \quad \neq \quad r = 010\_\ldots$

* Choose the i-th bit of r by flipping the i-th bit of $f(i)$.

∎

Lemma-C: The set of all languages $\mathcal{L}$ is uncountable.

Proof for Lemma-C: To prove this, we have to build a correspondence between $\mathbb{B}$ and $\mathcal{L}$, i.e., the two sets have the same size.

Recall that a language is a collection of strings from set Σ^*. We can represent the strings that are members of language $A \in \mathcal{L}$ as an infinite binary sequence χ_A (also called characteristic sequence of A) where its i-th bit takes value 1 if the i-th string of LS_{Σ^*} is in language A and value 0 if i-th string of LS_{Σ^*} is not in language A.

LS_{Σ^*}:	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc	…
$A = \{$		b,			ab,			bb,	bc,				… $\}$
$\chi_A =$	0	1	0	0	1	0	0	1	1	0	0	0	

Given a fixed LS_{Σ^*}, each language in $\mathcal{L}$ has a unique characteristic sequence of A.
The function $f: \mathcal{L} \rightarrow \mathbb{B}$, where $f(A)$ is the characteristic sequence of A is one-to-one and onto, and hence is a correspondence.
Thus, since $\mathbb{B}$ is uncountable, $\mathcal{L}$ is uncountable as well.

Proof for 4.18: Each Turing machine can recognize a single language.
From Lemma-C the set of all languages is uncountable, while from Corollary-A, the set of all Turing machines is countable.
Since there are uncountably many languages and countably many TMs, we conclude that some languages are not recognized by any TM. ∎

* The key idea is that the description of a TM must be a finite string whereas the content of a language can be represented by an infinite sequence.
This asymmetry is the reason that there are languages not recognized by a TM.