

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

TURING MACHINES

DISCRETE MATHEMATICS AND THEORY 2

MARK FLORYAN

GOALS!

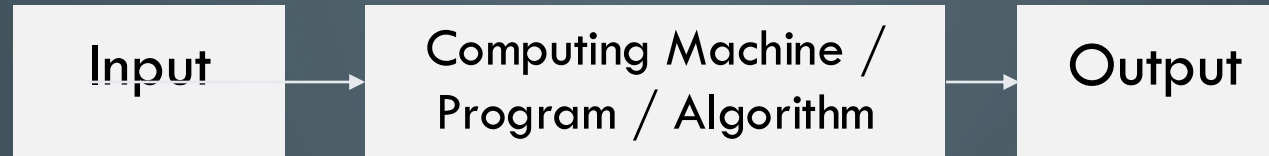
1. Our next computational model: The Turing Machine!
2. Analysis of different variations of Turing Machines (including non-deterministic Turing Machines).
3. Formalizing the concept of decidability versus recognizability, algorithms, and an introduction to computability theory.

The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural network connections. These elements consist of thin lines that branch out and terminate in small circles, creating a symmetrical, abstract pattern in each corner.

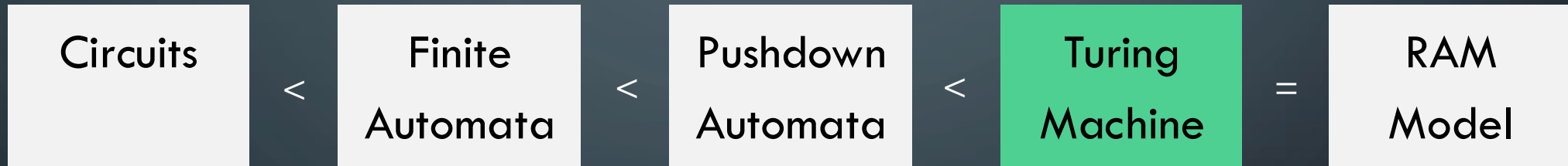
PART 1: REMINDER OF WHERE WE ARE / CHOMSKY HIERARCHY

OVERVIEW OF THEORY OF COMPUTATION

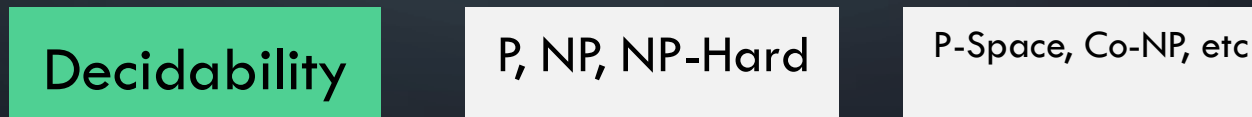
Defining Computation



Computational Models



Computational Complexity



WHAT IS A TURING MACHINE?

A **Turing Machine** (TM), sometimes called a **Deterministic Turing Machine** (DTM) is a finite state machine that can read/write from an infinite tape (memory)

Some other features of the Turing Machine:

1. A TM can both read and write to/from the tape
2. The TM contains a head that can move left and right along the tape
3. The tape is infinite
4. The special states for accepting / rejecting take effect immediately

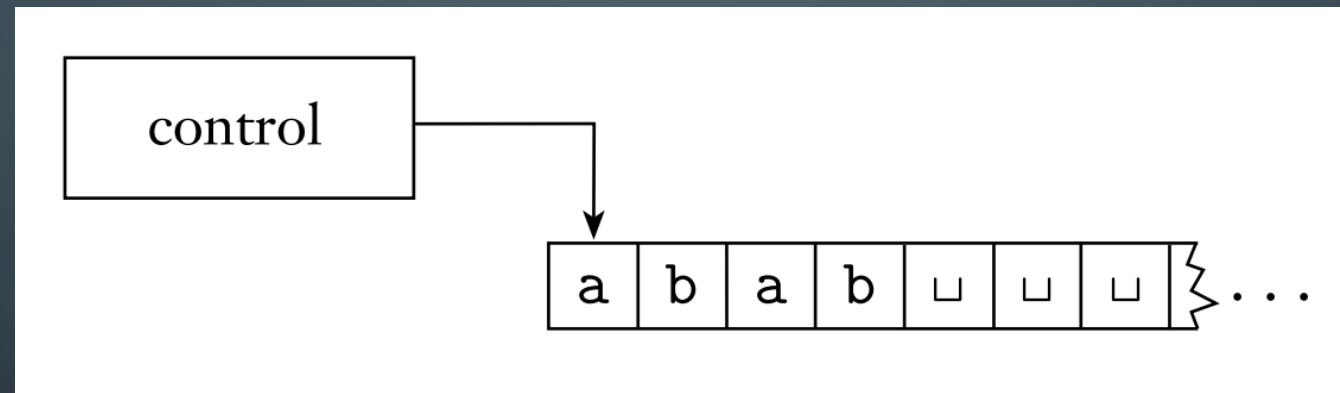
The background is a dark blue gradient. In the corners, there are white line-art illustrations of circuit boards or neural networks, with lines connecting to small circles.

INTRODUCING THE TURING MACHINE

WHAT IS A TURING MACHINE?

This control is a traditional DFA, except the accept/reject states take effect immediately.

The arrow here represents the head of the machine. It can move left and right and also read/write the symbol at that position.



This tape contains the input when execution begins.

EXAMPLE TURING MACHINE

Let's design a Turing Machine to recognize the following language:

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Any ideas on how we might do this?

Remember that the proposed string is written on the tape at the beginning of execution, and the head of the tape begins at index 0

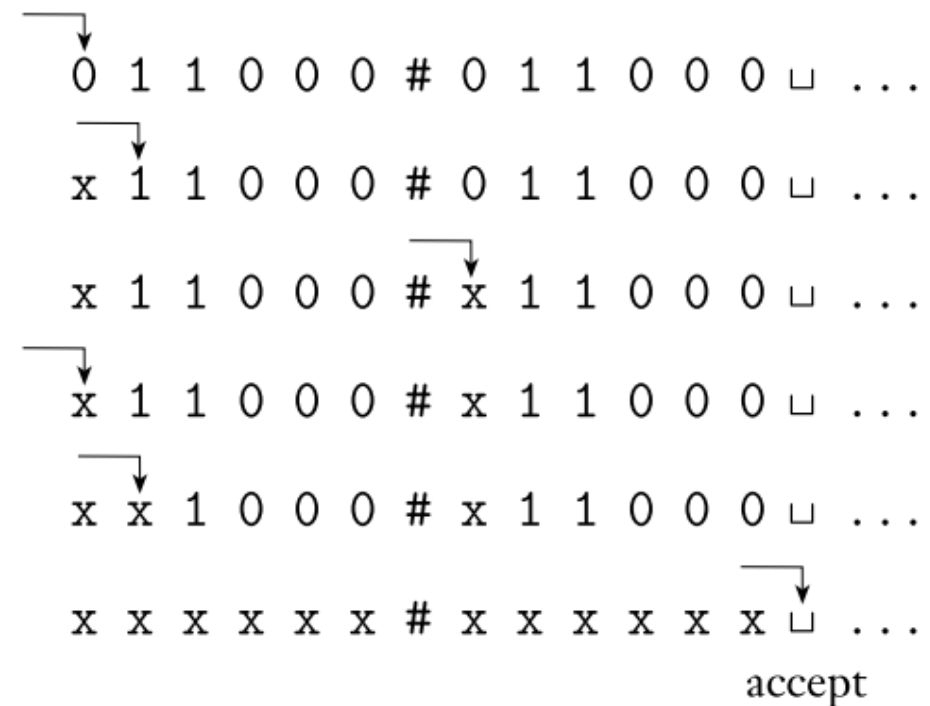
EXAMPLE TURING MACHINE

Let's design a Turing Machine to recognize the following language:

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Machine M_1 will do the following:

1. Zig-zag across the tape to corresponding positions on either side of the $\#$ symbol to check whether these positions contain the same symbol. If they do not or no $\#$ is found, reject.
2. When all symbols to the left of the $\#$ have been crossed off, check for any remaining symbols to the right of the $\#$. If any exists, reject otherwise accept.



FORMAL DEFINITION OF TM

A **Turing Machine** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, 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. $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{acc} \in Q$ is the accept state
7. $q_{rej} \in Q$ is the reject state, where $q_{rej} \neq q_{acc}$

Blank symbol often used to mark special cases, end of input, etc.

*L, R, S here represent moving the head **left** or **right** or **staying still***

Note that TMs have one accept and one reject state, and they cannot be equal.

TRANSITION FUNCTION

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ is the transition function

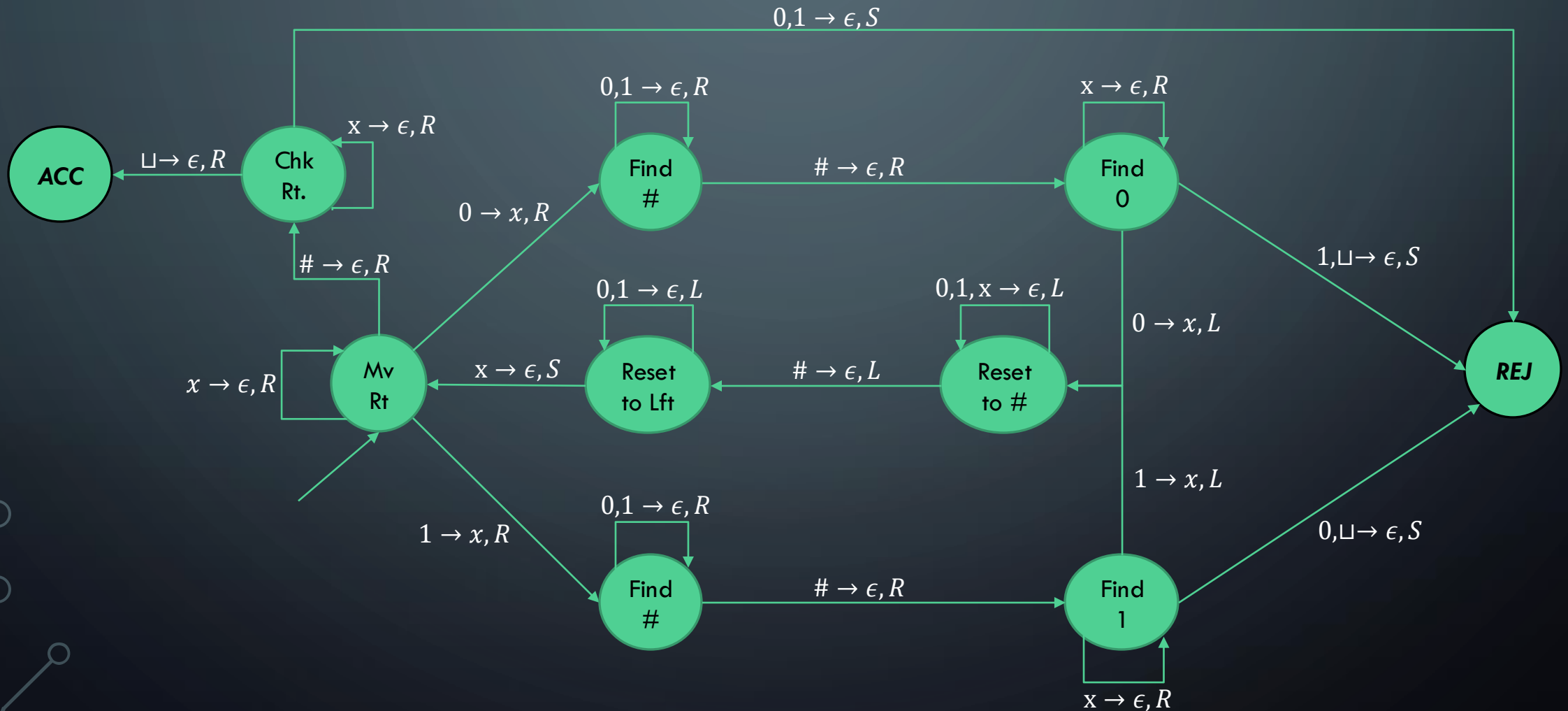
Where the TM transitions depends on this input. What state the machine is in and what symbol is currently on the tape at the head's current position.

Machine will enter a new state (optional)
Machine will write something to the tape (optional)
Head will move Left or Right (or S for staying put)

EXAMPLE TURING MACHINE

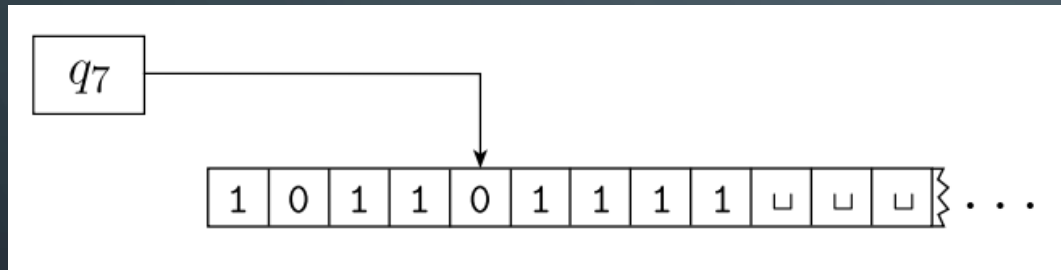
Let's design a Turing Machine to recognize the following language:

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$



CONFIGURATIONS OF A TM

A **configuration** of a Turing Machine is the complete state the machine is in at any point during execution. This includes the state, the contents of the tape, and the position of the head.



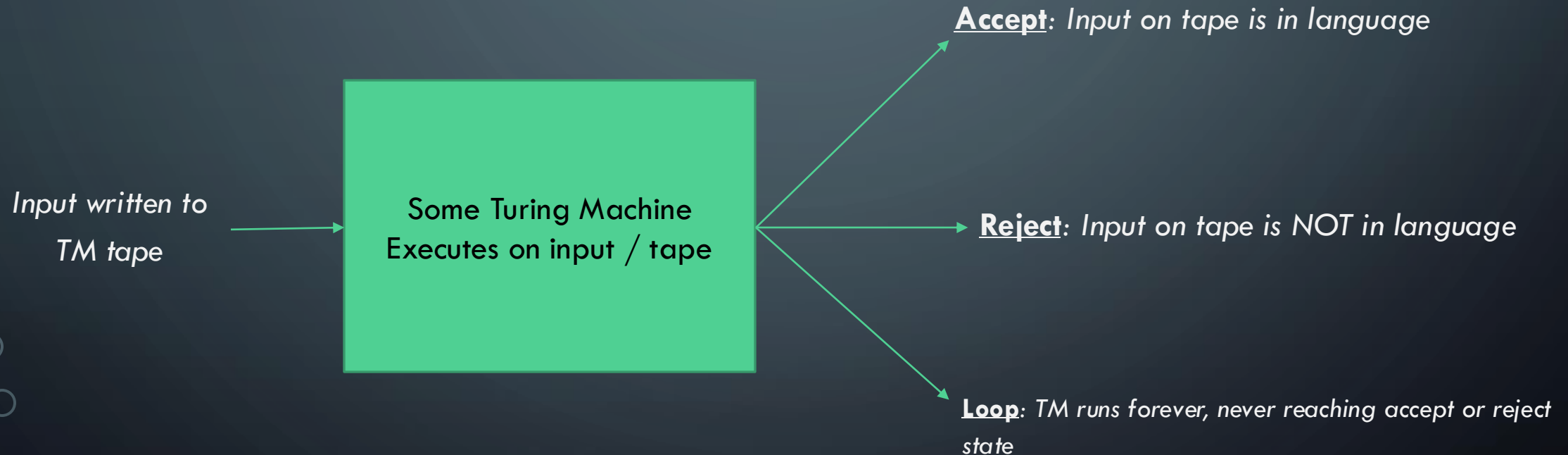
This machine is in state q_7 , and the contents of the tape / position of the head can be seen in the diagram.

A configuration of a TM can be represented succinctly as a string:

$1011q_701111$

RECOGNIZING VS DECIDING

When a Turing Machine executes, there are three possible outcomes



RECOGNIZING VS DECIDING

A Turing Machine **decides a language (is a decider)** if it never loops and always correctly accepts or rejects strings for the given language.

Some Turing Machine
Executes on input / tape

Accept: *Input on tape is in language*

Reject: *Input on tape is NOT in language*

Loop: *TM runs forever, never reaching accept or reject state*

A TM **decides the language** if it always halts and outputs one of these two possibilities. The language of this TM is said to be a **decidable language**.

RECOGNIZING VS DECIDING

A Turing Machine recognizes a language (is a recognizer) if it always accepts strings that are in the language, but might reject or might loop forever on strings that are NOT in the language.

Some Turing Machine
Executes on input / tape

Accept: Input on tape is in language

A recognizer will always accept when the string IS in the language

Reject: Input on tape is NOT in language

However, when the input string is NOT in the language, a recognizer might reject or it might just loop forever.

Loop: TM runs forever, never reaching accept or reject state

Languages that can be recognized are called Turing-Recognizable

The background is a dark blue gradient. In the corners, there are white line-art illustrations of circuit boards or neural networks, with lines connecting to small circles.

EXAMPLES: DESIGNING TURING MACHINES

PRACTICE 1: DESIGN A TM

Construct a Turing Machine that decides the following language:

$$A = \{0^{2^n} \mid n \geq 0\}$$

PRACTICE 1: DESIGN A TM

Construct a Turing Machine that decides the following language:

$$A = \{0^{2^n} \mid n \geq 0\}$$

Overall Approach, on input 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 end of the tape
5. Go to stage 1

PRACTICE 1: DESIGN A TM

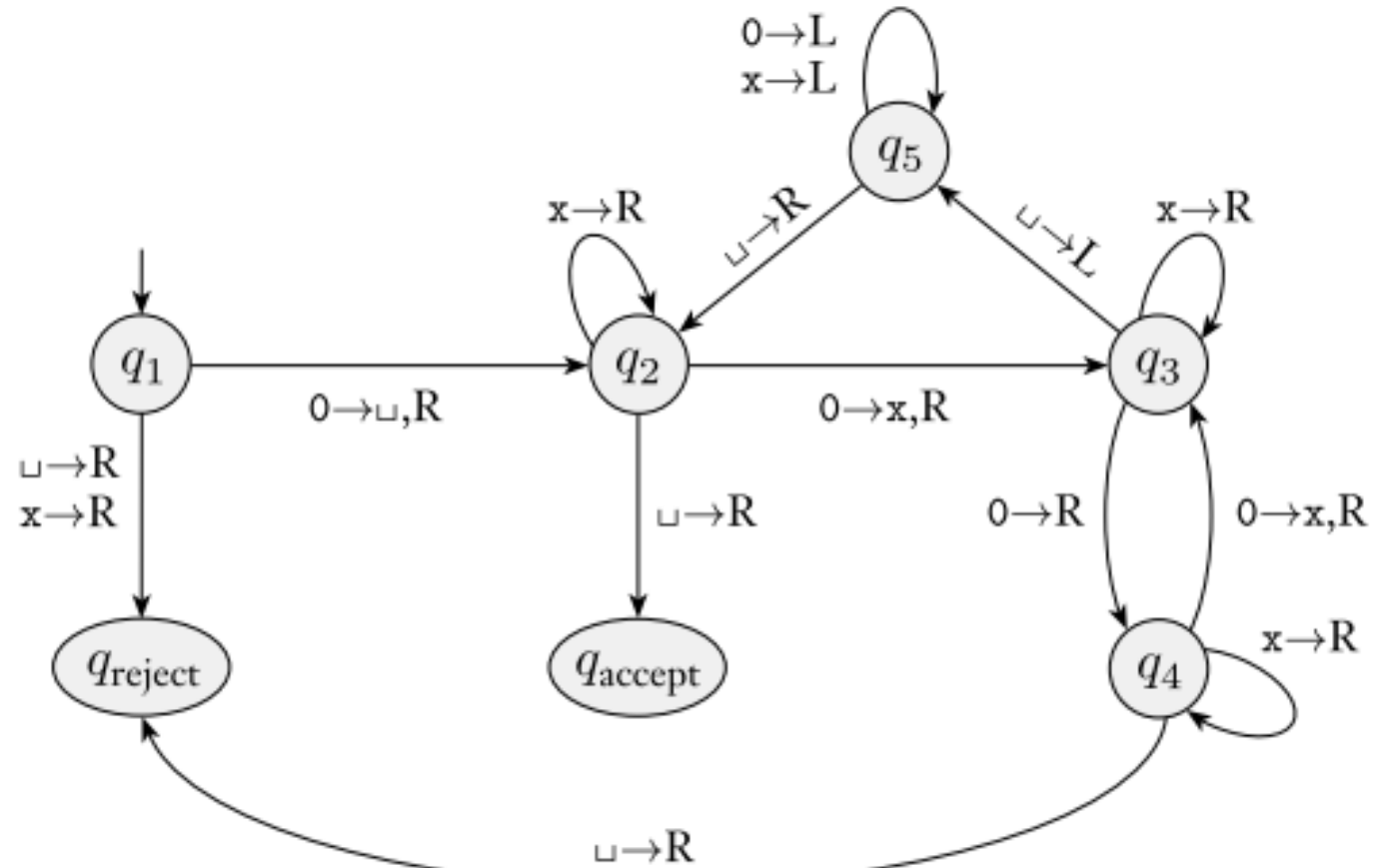
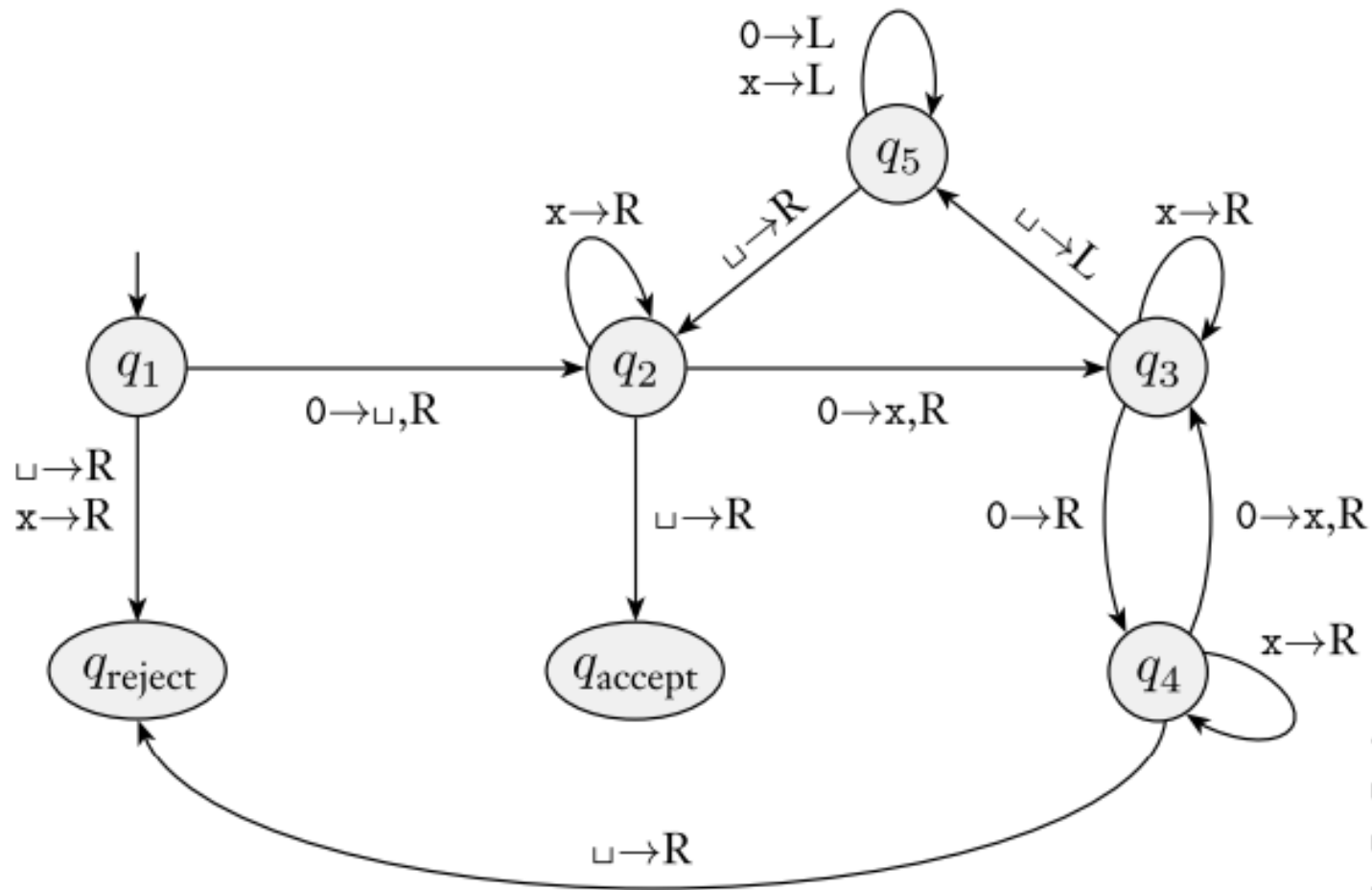


FIGURE 3.8
State diagram for Turing machine M_2

PRACTICE 1: DESIGN A TM



Q1 represents start state. We mark first 0 with blank
 Q2 represents we have seen one 0
 Q3 represents seen one 0 or even number of 0s
 Q4 represents seen an odd number of 0s
 Q5 represents moving head back to the front

$q_1 0000$	$\sqcup q_5 x 0 x \sqcup$	$\sqcup x q_5 x x \sqcup$
$\sqcup q_2 000$	$q_5 \sqcup x 0 x \sqcup$	$\sqcup q_5 x x x \sqcup$
$\sqcup x q_3 00$	$\sqcup q_2 x 0 x \sqcup$	$q_5 \sqcup x x x \sqcup$
$\sqcup x 0 q_4 0$	$\sqcup x q_2 0 x \sqcup$	$\sqcup q_2 x x x \sqcup$
$\sqcup x 0 x q_3 \sqcup$	$\sqcup x x q_3 x \sqcup$	$\sqcup x q_2 x x \sqcup$
$\sqcup x 0 q_5 x \sqcup$	$\sqcup x x x q_3 \sqcup$	$\sqcup x x q_2 x \sqcup$
$\sqcup x q_5 0 x \sqcup$	$\sqcup x x q_5 x \sqcup$	$\sqcup x x x q_2 \sqcup$
		$\sqcup x x x \sqcup q_{\text{accept}}$

EXAMPLE 2!!

Construct a Turing Machine that decides the following language:

$$\mathcal{C} = \{a^i b^j c^k \mid i \times j = k \wedge i, j, k \geq 1\}$$

EXAMPLE 2!!

Construct a Turing Machine that decides the following language:

$$C = \{a^i b^j c^k \mid i \times j = k \wedge i, j, k \geq 1\}$$

Overall Idea:

On input String w :

1. Scan the tape to make sure the input is in the form $a^+ b^+ c^+$, if not reject.
2. Return the head to the left end of the tape.
3. Cross off the first a and scan to the right until first b is found. Shuttle between crossing off one b and scanning right to cross off one c until all the b 's are gone. If all c 's are crossed out, and b 's remain, reject.
4. Restore the crossed off b 's and repeat stage 3 if there is another a . If all a 's are crossed off, determine whether all c 's are crossed off. If yes, accept, otherwise reject.

EXAMPLE 3!!

Element Distinctness Problem: Can we decide the language:

$$E = \{\#x_1\#x_2\# \dots \#x_n \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

EXAMPLE 3!!

Element Distinctness Problem: Can we decide the language:

$$E = \{\#x_1\#x_2\# \dots \#x_n \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

Key ideas we will need for this one:

1. The **tape can be marked to keep track of loops** (which characters are being compared with which characters). This is why the **#** symbols are useful.
2. How do we actually **compare the characters**? It is annoying but can be done with many states. How do you think that would work?

TRACKING LOOPS!!

How do we keep track of loops with a Turing Machine?

First, introduce new tape symbols that represent the loop beginning and end. Here we will use (and). Remember that goal is to compare every pair of characters.

#	f	(a	#	c	#	e	#	s)	l	#	k	␣
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	------

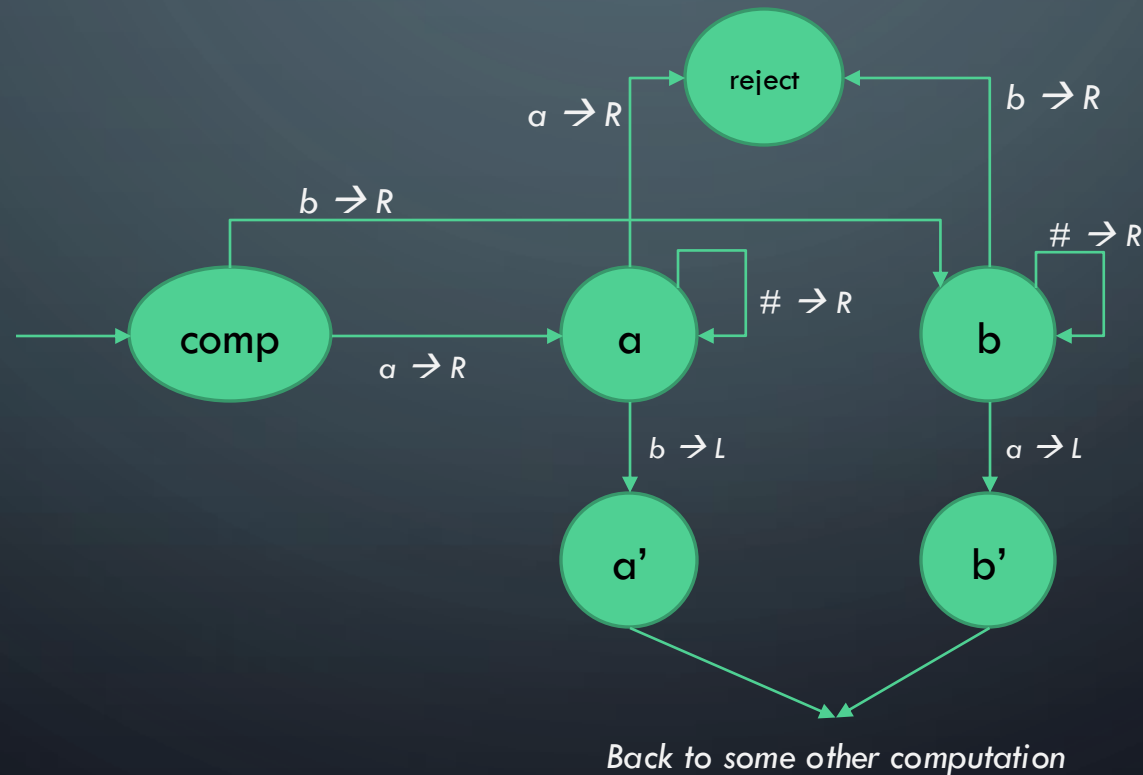
When starting an outer loop, mark the relevant part of the tape with open paren.

Mark closing paren to mark outer loop location if necessary.

COMPARING CHARACTERS

How hard is it to compare characters?

This machine will have to check characters for equivalence. How do we do this? Let's suppose that $\Sigma = \{a, b\}$ and tape is pointing at left side of $c_1\#c_2$



EXAMPLE 3!!

Element Distinctness Problem: Can we decide the language:

$$E = \{\#x_1\#x_2\# \dots \#x_n \mid \text{each } x_i \in \Sigma \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

On input w :

1. Place a mark on top of the leftmost tape symbol. If that symbol was blank, accept. If that symbol was $\#$, continue with the next stage. Otherwise, reject.
2. Scan right to the next $\#$ and place a second mark on top of it. If no $\#$ is encountered before a blank symbol, only x_1 was present, so accept.
3. By zig-zagging, compare the two strings to the inside of the marked $\#$ s. If equal, reject.
4. Move the rightmost of the two marks to the next $\#$ symbol to the right. If no $\#$ symbol is encountered before a blank, move the leftmost mark to the next $\#$ to its right and the rightmost mark to the $\#$ after that. This time, if no $\#$ is available for the rightmost mark, all the strings have been compared, so accept.
5. Loop: Go to step 3.

The background is a dark blue gradient with faint, large concentric circles. In the corners, there are white line-art illustrations of circuit boards or neural networks, featuring lines and small circles.

TURING MACHINE VARIANTS

MOTIVATING QUESTION

Can we have different features of TMs that increase / or don't the recognizing power of the traditional TM?

Some we will see:

Turing machines with multiple tapes

Non-deterministic Turing Machines

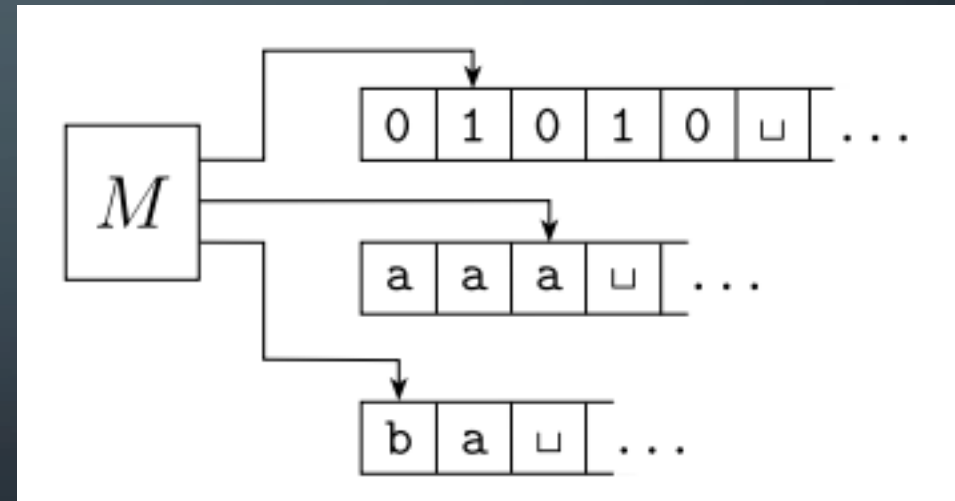
MULTITAPE TURING MACHINE

A **Multitape Turing Machine** is like an ordinary TM but it has several tapes instead of one. Each tape has an independent head that can be moved.

The transition function is updated to:

$$\begin{aligned} \delta: Q \times \Gamma^k \\ \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k \end{aligned}$$

Here, k is the number of tapes. So each tape can be read at each step of computation, each head can write / move as well.



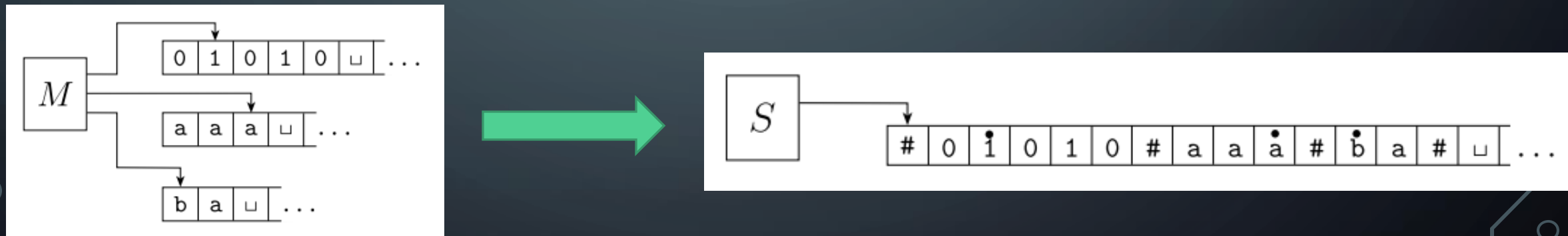
MULTITAPE TURING MACHINE

Theorem: Every Multitape Turing Machine has an equivalent single-tape Turing machine.

MULTITAPE TURING MACHINE

Theorem: Every Multitape Turing Machine has an equivalent single-tape Turing machine.

Goal: Given an arbitrary MTTM, convert it into an equivalent TM



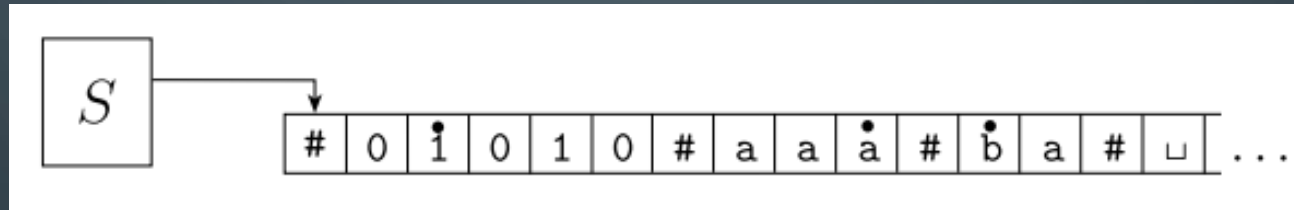
***Need to argue that S simulates M's computation exactly in all scenarios*

MULTITAPE TURING MACHINE

Two major issues we need to overcome for this proof:

Issue 1: How to simulate multiple tapes / heads with only one tape / head?

Issue 2: How to handle space constraints fitting k tapes on just 1?

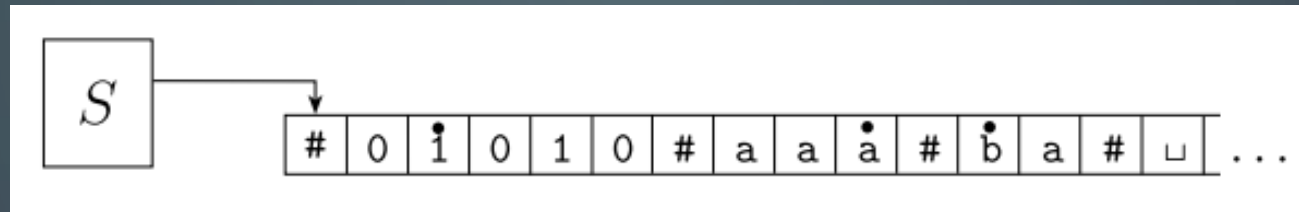


For issue 1, we make special versions of each tape symbol that represent a “virtual head” being positioned at that location.

For issue 2, we separate each tape by a special # symbol and create a subroutine that shifts the contents of everything to the right down when we need more space on any individual tape.

MULTITAPE TURING MACHINE

Summary of how to simulate M with S



$S =$ on input $W = w_1w_2w_3 \dots w_n$:

1. First S puts its tape into the format representing all k tapes of M , separated by the $\#$ symbol. The special “virtual head” symbol is used at the left most symbol on each of the k “sections” of the tape.
2. To simulate a single step, S first steps across the whole tape to see which symbols are under each of the virtual heads (reading each of the tapes), then S simulates writing and updating the “head” of the tape for each individual section according to M ’s original transition function.
3. If at any point S runs out of space on any tape, we run a subroutine that shifts everything on the tape over by one, creating one more cell of space on this particular tape.

MULTITAPE TURING MACHINE

Theorem: Every Multitape Turing Machine has an equivalent single-tape Turing machine.

Corollary: A language is Turing-recognizable iff some Multitape Turing Machine recognizes it.

Proof:

Direction 1: A Turing recognizable language is recognized by some Turing Machine, and any TM is also a valid multitape TM.

Direction 2: If a multitape TM recognizes a language, then it can be converted into an equivalent single tape TM as described earlier.

NON-DETERMINISTIC TURING MACHINES (NTM)

A non-deterministic Turing Machine (NTM) can branch into many possibilities.

Transition function has form: $\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, S\})$

The tape, state, etc. are all branched to an independent machine. No resources are shared across the branches.

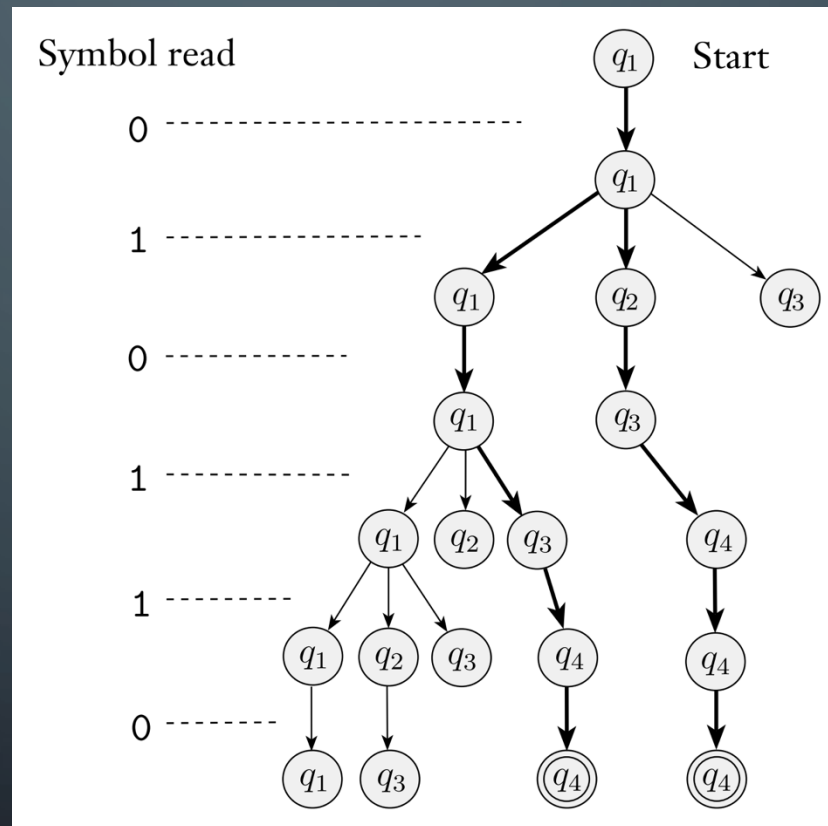
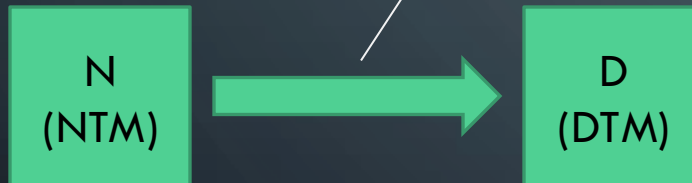
NTM VERSUS DTM

Theorem: Every NTM has an equivalent DTM

NTM VERSUS DTM

Theorem: Every NTM has an equivalent DTM

Using D , simulate N 's execution exactly



Idea: Machine N has branching computations. Let's search this computation tree in a breadth-first manner.

Issues:

How to switch between branches?

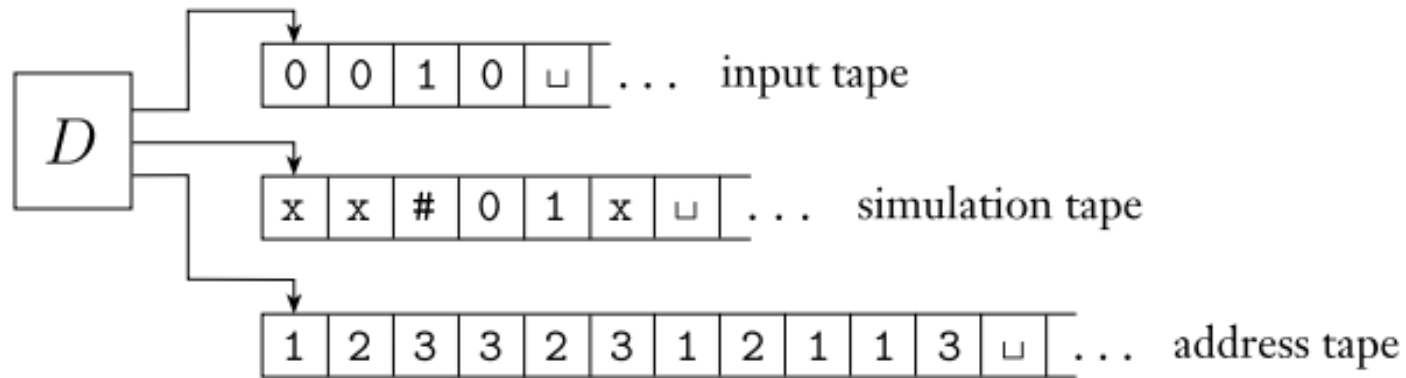
Why BFS instead of DFS?

Keeping track of tree location?

NTM VERSUS DTM

Theorem: Every NTM has an equivalent DTM

Proof Idea: Simulate the NTM with a DTM (multi-tape)



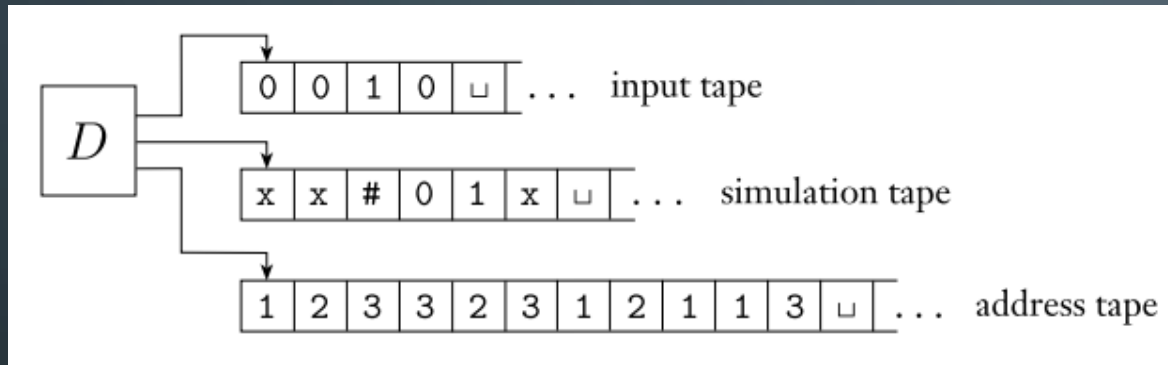
Tape 1 contains the input and will never be altered.

Tape 2 contains N 's tape at that particular branch of execution as we simulate

Tape 3 is the address tape, it tells us which branch of computation to simulate at each step (more about this on the next slide).

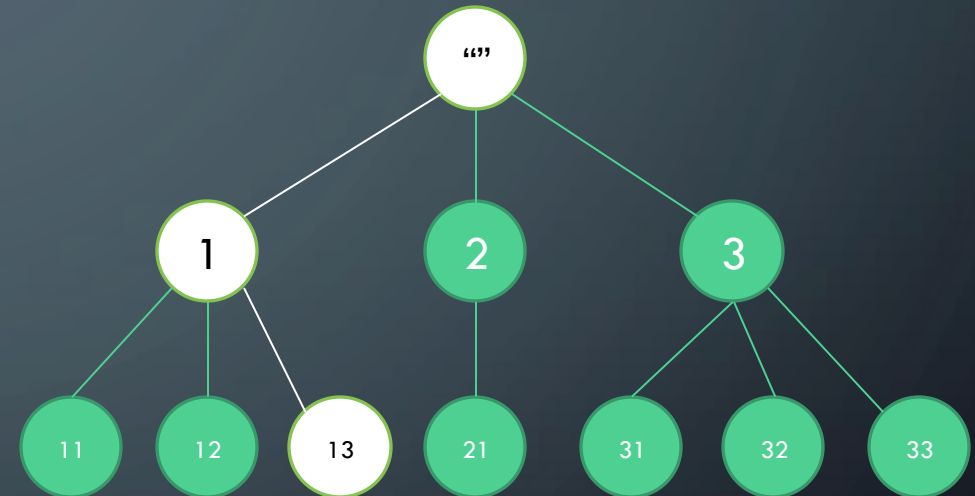
NTM VERSUS DTM

Theorem: Every NTM has an equivalent DTM



Let's look at Tape 3 for a moment

Tape 3 tells you which direction down tree to go each time you make a step in your computation. Each cell is at most b , where b is the max number of branches possible for that machine.

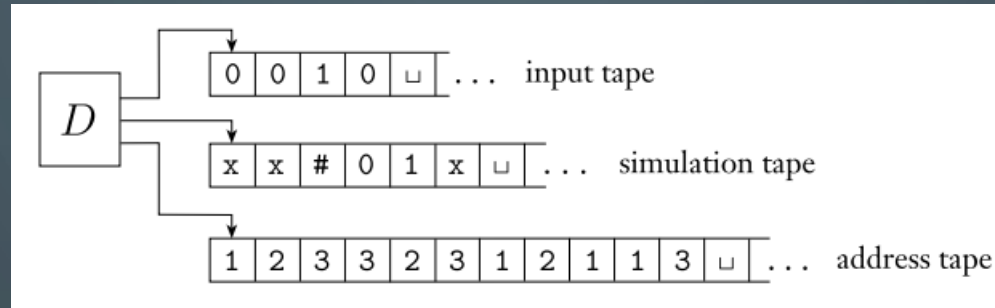


For example, if tape 3 contains "13", we will simulate the highlighted (white) path above and terminate (even if there is more computation possible).

Tape 3 will try "", 1, 2, 3, 11, 12, 13, 21, 31, 32, 33,

NTM VERSUS DTM

Theorem: Every NTM has an equivalent DTM



Given Machine N, Machine D Proceeds As Follows:

1. Tape 1 contains the input, tapes 2 and 3 are empty.
2. Copy tape 1 onto tape 2 to initialize the simulation.
3. Use tape 2 to simulate N on input. At every stage of computation, look at tape 3 to determine which non-deterministic branch to follow (which next state out of many choices to take). If we reach an accept state, then accept! If we run out of branches to follow on tape 3, or we reach a reject state, or this configuration is invalid (the choice on tape 3 is not a valid choice), then abort this branch and go to step 4.
4. Replace the string on tape 3 with the next string in the string ordering (e.g., "11" might become "12"). Simulate the next branch by going to step 2.
5. This machine rejects if all branches are enumerated and no accept state is found (note the machine could loop forever).

NTM VERSUS DTM

Theorem: A language is decidable if and only if some non-deterministic Turing machine decides it (same holds for recognizing).

This follows from our previous proof. Any NTM can be simulated with a DTM so these machines are equivalent for deciding languages!

LAST CONCLUSION

Any computational model satisfying reasonable requirements to that of a TM is equivalent in power to a TM

Most notably:

1. Access to unlimited memory
2. The ability to perform a finite amount of work in a single step

The background is a dark blue gradient. In the corners, there are white, stylized circuit-like lines. These lines consist of straight segments and small circles, resembling a network or a circuit board layout. They are positioned in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

THE DEFINITION OF AN ALGORITHM

SHORT DISCUSSION

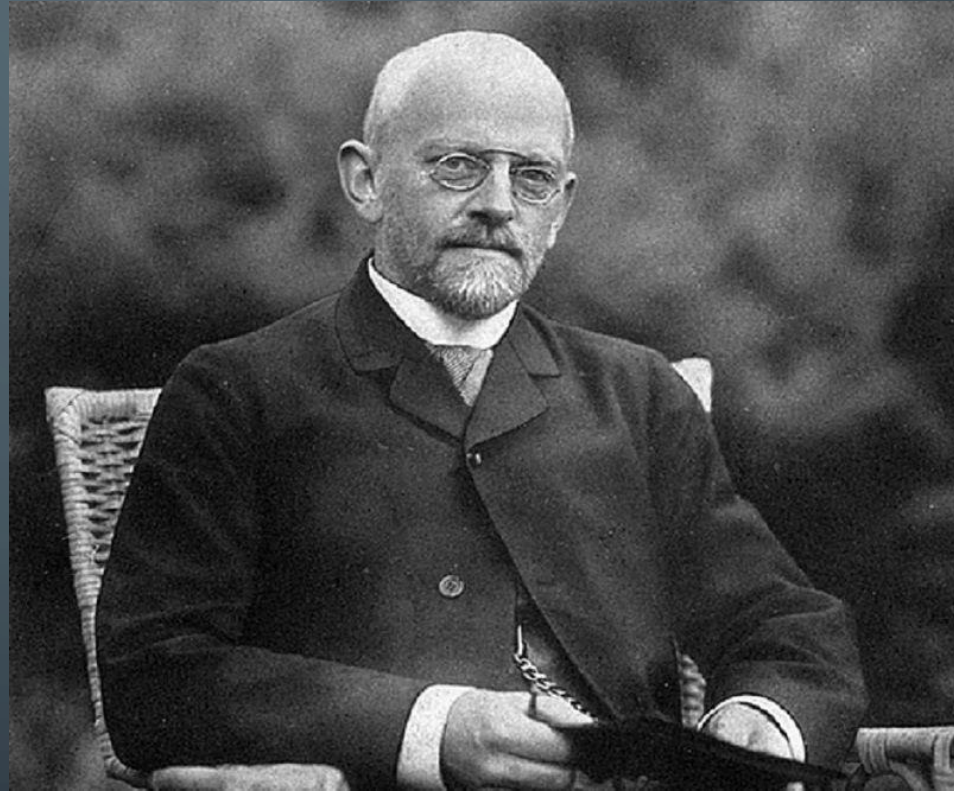
What is an **Algorithm**? Does seeing the Turing Machine change your perspective on this at all?

***It was not until the 20th century that the notion of an algorithm was defined precisely.*

HILBERT'S PROBLEMS

David Hilbert: German Mathematician

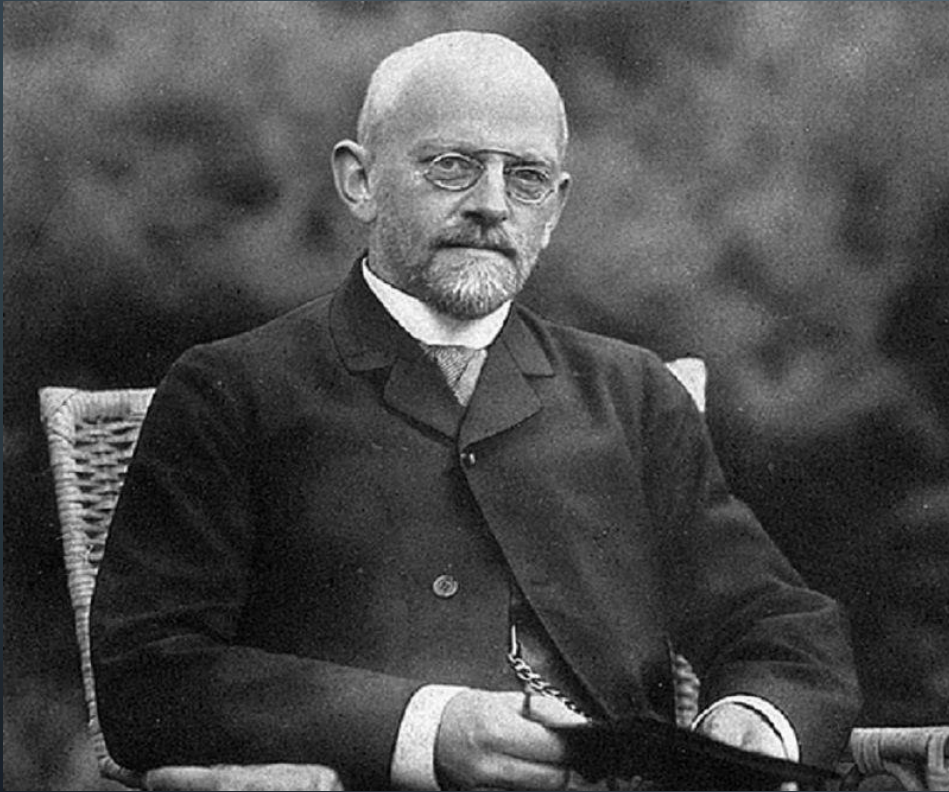
In 1900, Hilbert gave a now famous address at the International Congress of Mathematicians in Paris. He identified 23 mathematical problems and posed them as a challenge for the coming century. Hilbert's 10th problem concerned algorithms.



What was Hilbert's 10th problem: *Devise an algorithm that tests whether a polynomial has an integral root (he did not use the word "algorithm").*

HILBERT'S PROBLEMS

David Hilbert: German Mathematician



Hilbert used the word “devise”, implying that some algorithm / process must exist (he was wrong about that part).

The 10th problem is unsolvable, but mathematicians in 1900 only had an intuitive understanding of algorithm.

The world needed a formal definition of an “algorithm” in order to prove that this particular problem was unsolvable...

THE CHURCH-TURING THESIS

The Church-Turing Thesis (1936): Lambda-Calculus (Alonzo Church) and Turing Machines (Alan Turing) provide the mechanism for formally defining an algorithm.

So...an algorithm is any process that can be programmed on a Turing Machine in order to recognize or decide some function (or language).

*Intuitive notion
of algorithms*

equals

*Turing machine
algorithms*

FIGURE 3.22
The Church-Turing thesis

FINDING POLYNOMIAL ROOTS IS RECOGNIZABLE

Problem: Can you recognize (not decide) the following language:

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}$$

Also recall that this language is recognizable if you can find the root if it exists, but you can loop forever if it does not exist.

Note that the polynomial is guaranteed to be over just one variable.

FINDING POLYNOMIAL ROOTS IS RECOGNIZABLE

Problem: Can you recognize (not decide) the following language:

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}$$

Solution:

On input $\langle p \rangle$:

1. Evaluate p with x set successively to the values $0, 1, -1, 2, -2, 3, -3, \dots$
2. **Accept** if at any point the polynomial evaluates to 0

Notice that this solution loops forever anytime the answer should be “reject”. It is possible to turn this into a decider (see book for details)

FINDING POLYNOMIAL ROOTS IS RECOGNIZABLE

Problem: Can you recognize (not decide) the following language:

$$D_1 = \{p \mid p \text{ is a polynomial over } \{x, y, z \dots\} \text{ with an integral root}\}$$

This problem is undecidable, meaning there does not exist an algorithm (Turing Machine) that decides it. Now that we understand Turing Machines, we can begin to formulate how some problems cannot be decided.

WHAT DID WE LEARN IN THIS DECK

1. Definition of Turing Machines, both deterministic and non-deterministic along with other variants
2. Simple design of algorithms using Turing Machines
3. Definitions of recognizable vs decidable languages
4. Practice with designing Turing Machines for simple problems.