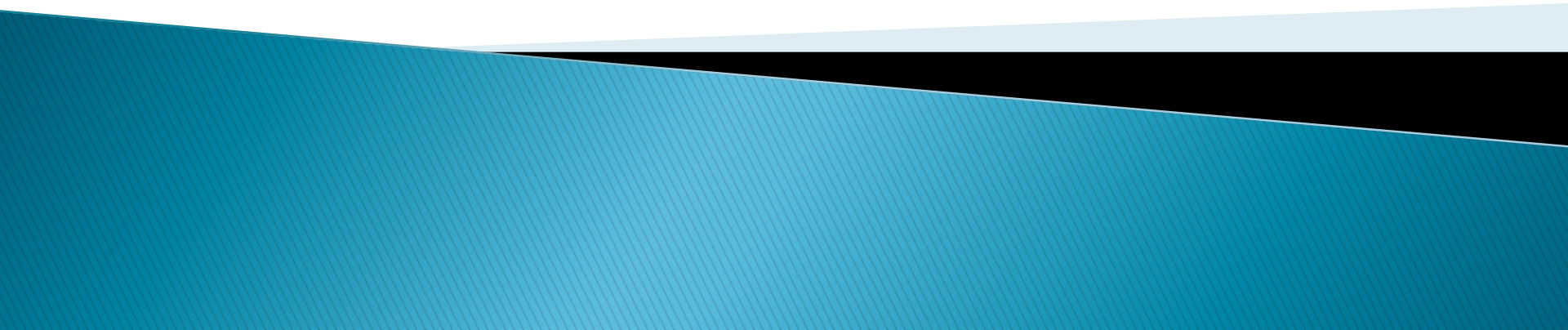



# Turing Machines

TM Variants



# Turing Machine

- ▶ A Turing Machine consists of:
    - A finite state machine
    - An input tape
    - A movable read/write tape head
  - ▶ A move of a Turing Machine
    - Read the symbol on the tape at the current position of the tape head
    - Change the symbol on the tape at the current position of the tape head
    - Move the tape head
    - Change the state of the machine based on current state and symbol read
- 

# Turing Machine

- ▶ A note about the book's definition
  - TMs are deterministic
  - Each input transitions to one (not multiple) output
  - No  $\epsilon$ -transitions
- That said, for all states where transitions are not defined:
  - it is assumed there is a transition to the reject state.


# Turing Machine

- ▶ Running a Turing Machine
  - The execution of a TM can result in 3 possible cases:
    - Accept
      - The machine halts in the accepting state
    - Non-accept
      - The machine halts in the rejecting state
      - The machine goes into an “infinite loop”

# Turing Machine

- ▶ Any language that is recognized by some Turing Machine is said to be Turing-recognizable.
  - The Turing Machine may loop on some rejected strings
  - Sometime termed recursively enumerable in other books.
- ▶ A language is Turing-decidable (or just decidable) if some Turing Machine decides it
  - It is recognized by a Turing Machine that halts on all input
    - No infinite loops
  - Sometimes termed a recursive language in other books.

# Context Switch

- ▶ At this point, we are going to leave behind the formal description of Turing Machines.
  - ▶ We will use implementation and high-level descriptions to talk about what we want a Turing Machine to do.
  - ▶ Concepts will be more abstract.
  - ▶ We'll be looking more big picture at what Turing Machines can do. What type of problems can we solve algorithmically.
- 

# Algorithms and Turing Machines

- ▶ Different levels for describing algorithms:
  - Formal Description
    - Define the states and transitions of a TM
  - Implementation Description
    - English prose that describes how the TM works.
  - High-Level Description
    - English prose that describes the algorithm (no mention of TM implementation).

# Turing Machine Properties

- ▶ There are many equivalent ways to define a TM
  - Each equivalent representation recognizes the same class of languages and decides the same class of languages
  - We'll conclude that adding bells and whistles to our basic TM won't give it any extra problem-solving power
- ▶ Today we will look at a few that the book mentions (and doesn't mention):
  - TM with STAY option
  - TMs with a 2-way infinite tape
  - TMs with multiple tapes / heads
  - Non-Deterministic TMs



# Alternative TM Definitions

- ▶ Turing Machines with a “STAY” option
  - Allows the tape head to stay where it is
  - $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, S\}$
- ▶ How could you simulate this feature with an original TM?
  - We can just move right and then back left
  - In more detail:
    - For any STAY case (and there can be only finite)
      - Create a new state that we will move to when we move to the right.
      - From that state, have it automatically transition back to the left and to the desired state on any tape symbol

# Alternative TM Definitions

- ▶ TM with a two-way infinite tape
- ▶ How could you simulate this feature with the original TM?
  - Two different possibilities
    - Map the two-way tape so that everything right of the start point maps to even locations on the one-way tape, and everything to the left of the start point maps to odd locations on the one-way tape
    - Ignore the possibility of going left of start. When it needs to happen, enter a dedicated set of states that shift everything on the tape one spot to the right, put in a blank at the left, then continue.

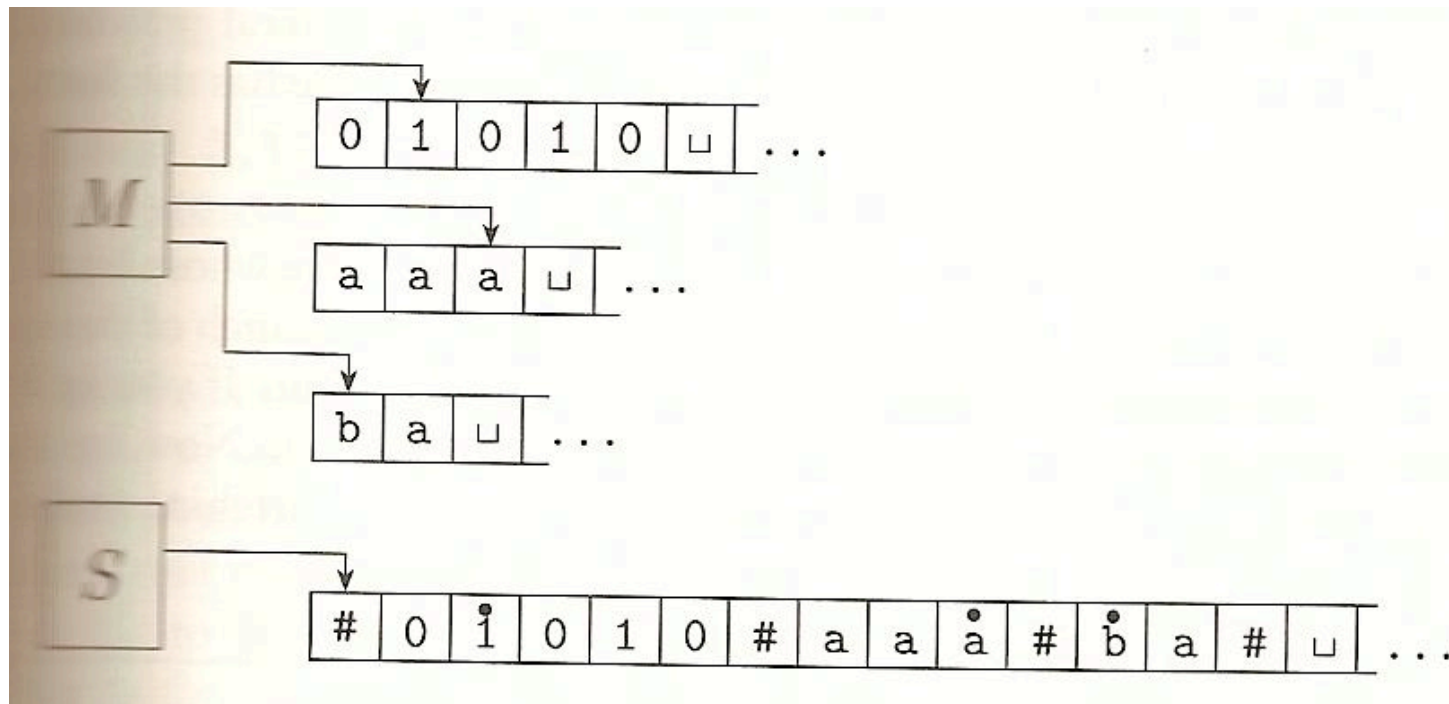
# Alternative TM Definitions

- ▶ TM with multiple tapes and heads
  - $\delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{R, L\}^n$
- ▶ How could you simulate this feature with the original TM?
  - Basic idea:
    - Define a special symbol on the tape that will delimit one of the multi-tapes from the next.
      - E.g. #00111#abba#ijk# using # to delimit the tapes
    - Use a marking (mark one symbol in each represented tape) to indicate the current location of each tape head
    - Shift things to the right as necessary to make room when a tape has to move beyond its current boundaries

# Aside – “Marking” with a TM

- ▶ For each symbol in the alphabet, have a corresponding “marked” symbol in the tape alphabet
- ▶ The machine can easily assess whether a given symbol is a marked symbol, and alter as desired
- ▶ The marking allows the original symbol to be recovered
  - It’s not the same as “crossing it out”

# TM with Multiple Tapes




# Non-Deterministic TM

- ▶ Basic TM in the book is deterministic
- ▶ Can define a non-deterministic TM
- ▶ Non-deterministic TM
  - Machine has a choice of moves
  - $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{R, L\})$
  - For a string  $w$ 
    - As long as there is one path that causes the TM to halt in the accept state,  $w$  is accepted.

# Non-Deterministic TM


- ▶ Theorem 3.16: Every non-deterministic TM has an equivalent deterministic TM

# Non-Deterministic TM

- ▶ Why they are equivalent
    - Given an NDTM, it is possible to construct a single multi-taped DTM that accepts the same language (and we now know this is equivalent to our original TM):
    - Basic idea:
      - Think of the NDTM as a tree with root node at the start.
      - At every move, the tree can split into 0 or more branches (there will be some maximum number of possible splits)
      - Have the DTM simulate all paths of the NDTM
- 



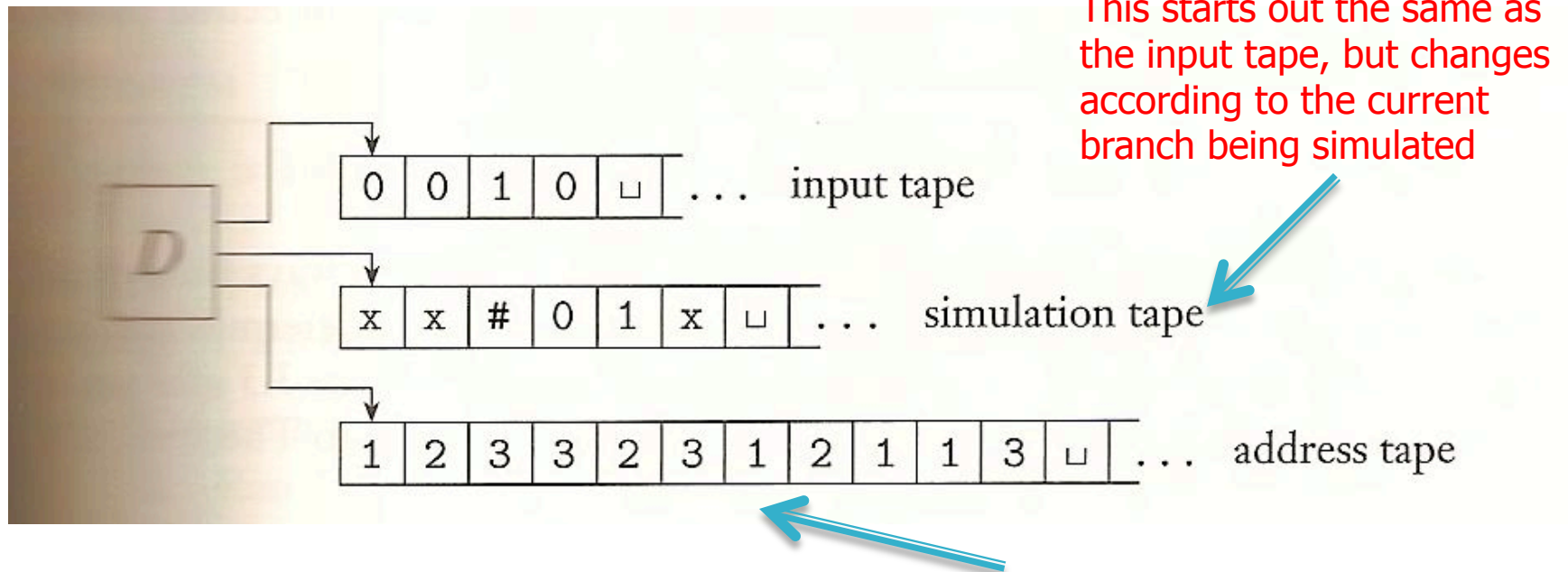
# Non-Deterministic TM

- ▶ If any of the NDTM paths halts and accepts, so too does the DTM
  - ▶ How should the DTM go about simulating all the paths (searching the tree)?
    - Breadth first search or
    - Depth first search
      - Breadth first search guarantees that if there is an accepting path, it will eventually find it, because each successive level of the tree has finite paths to consider.
      - Depth first search might wander down an infinitely looping branch and never make it to an existing accepting branch
- 

# Non-Deterministic TM

- ▶ Construction of DTM
- ▶ Use a 3 tape TM
  - Tape 1: input tape
    - Contains the input to be simulated
  - Tape 2: simulation tape
    - Simulates a branch of computation of the NDTM
  - Tape 3: address tape
    - Gives an encoding of the path from root to current node in the NDTM to simulate.

# Non-Deterministic TM



This starts out the same as the input tape, but changes according to the current branch being simulated

This indicates the sequence of branching moves to take for this branch, starting from the root (take the first branch out of the root; take the second branch from that node; ...)

# Non-Deterministic TM

- ▶ Running the machine:
  1. Initially tape 1 contains input, 2&3 empty
  2. Copy tape 1 to tape 2
  3. Simulate machine on tape 2...making non-deterministic choices based on sequence on tape 3. If accept, then accept
  4. Replace tape 3 with next “path” to consider and repeat steps 1–4.
- 5. If all paths reject (i.e. the NDTM always halts), then halt and reject.

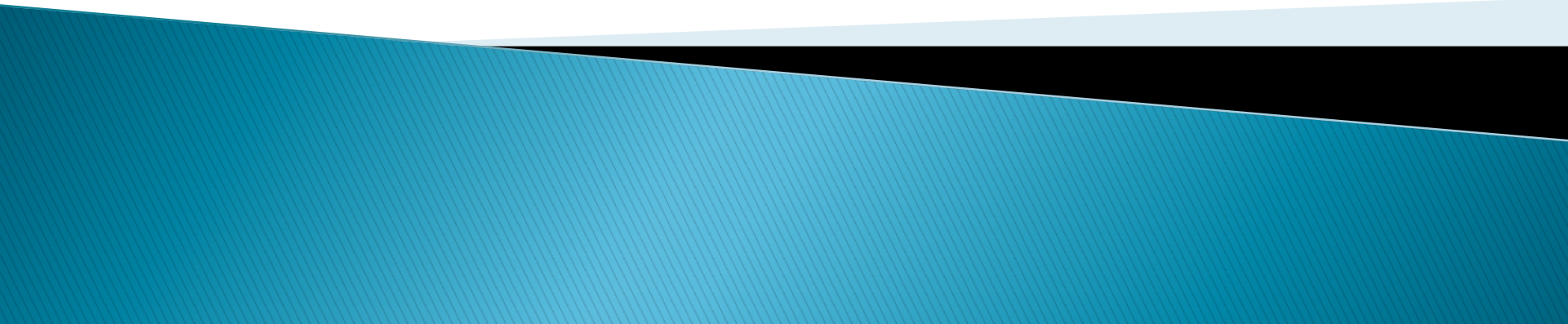
# Non-Deterministic TM

- ▶ Implications of NDTM
  - A language is Turing-recognizable if and only if some NDTM recognizes it
  - A language is decidable if and only if some NDTM decides it.

# Summary

- ▶ So in short...the following are equivalent:
  - Deterministic Turing Machines (our original)
  - Non-Deterministic Turing Machines
  - Turing Machines with a 2-way infinite tape
  - Turing Machines with multiple tapes / heads
  - Turing Machines with a STAY option

# Church–Turing Thesis



# The Church–Turing Thesis (1936)

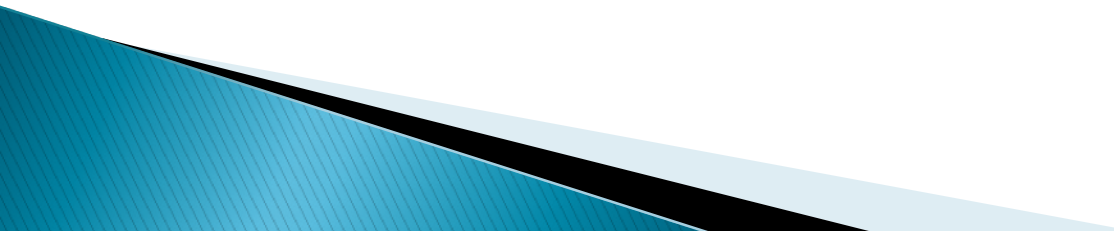
- ▶ “Any algorithmic procedure that can be carried out by a human or group of humans can be carried out by some Turing Machine”
  - Equating algorithm with running on a TM



# Algorithms

- ▶ An algorithm
  - Collection of instructions
  - Procedure
  - Recipe
- ▶ In computation, equivalent with running on a Turing machine.

# Algorithms and Turing Machines

- ▶ An algorithm always terminates
  - ▶ A semi-algorithm may go into an infinite loop on inputs that are not in the language
  - ▶ Church–Turing Thesis says that:
    - A language is algorithmically solvable if and only if it is Turing-decidable (all inputs halt)
    - A language has a semi-algorithm if and only if it is Turing-recognizable.
- 

# Algorithms and Turing Machines

- ▶ What does it mean to say that “a language is algorithmically solvable”
  - Language is a set of strings
  - We can cast our problems that we want to solve in terms of strings that are accepted or rejected
    - Frame the problem as a decision problem with a yes / no answer
    - Instances of the problem get encoded as strings
    - The language corresponds to encodings of the problem that have “yes” answers
    - The problem is algorithmically solvable if we can define a TM that correctly accepts only those strings that correspond to “yes” occurrences and rejects (without looping) all others

# Algorithms and Turing Machines

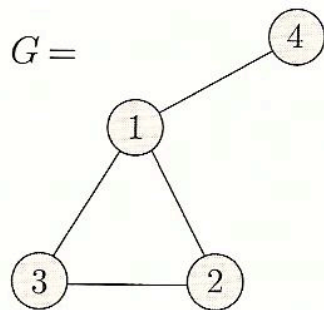
- ▶ Running algorithms on objects:
  - TMs take strings as input.
  - For running “algorithms” on other objects,
    - Must encode the object as a string.
    - Any decent encoding will do.
    - When running TMs on objects, it is assumed that decoding gets performed by the TM and that input is valid.
  - Notation:
    - If  $O$  is an object to be input to a TM,  $\langle O \rangle$  is the encoded object.
    - If  $O_1, O_2, \dots, O_n$  are multiple objects to be used as input to a TM,  $\langle O_1, O_2, \dots, O_n \rangle$  is the encoded list of objects.

# Example

- ▶ Let  $A$  be the set of strings representing undirected graphs that are connected.
  - In a connected graph, every node can be reached from every other node.
  - $A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}$

# Example

- First we need an encoding.



$\langle G \rangle =$

$(1, 2, 3, 4) ((1, 2), (2, 3), (3, 1), (1, 4))$

# Example

- ▶ High level description of a TM,  $M$ , that decides  $A$ :
- ▶ First thing  $M$  must do is check validity of format of the input. Reject if input is invalid. (This step is usually implicit in the TM description.)
- ▶ On input  $\langle G \rangle$ , where  $G$  is a graph:
  1. Select first node of  $G$  and mark it
  2. Repeat the following stage until no new nodes are marked:
  3. For each node in  $G$ , mark it if attached via an edge to a marked node.
  4. Scan nodes of  $G$  to see if all are marked. If yes, accept, if no, reject.

This is the notation we will use to represent high-level TM descriptions

# Note about Encoding as a String

- ▶ We encoded a graph as a string in the previous example for the TM to read as input
- ▶ In fact, strings can easily be used to represent:
  - Polynomials
  - Graphs
  - Grammars
  - Automata
  - Combinations of these things
- ▶ We'll be considering many of these objects as our starting point now