COM3064
Automata Theory

Week 11: Variants of Turing Machines

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli
CENG280 Formal Languages and Abstract Machines, Halit Oğuztüzün

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 moves to Left or Right.
  - is **deterministic**.

- There are 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**
  - **Turing Machines with Semi-Infinite Tape**
  - **OfflineTuring Machines**
  - **Multidimensional Turing Machines**

Variants of Turing Machines

- 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

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.**

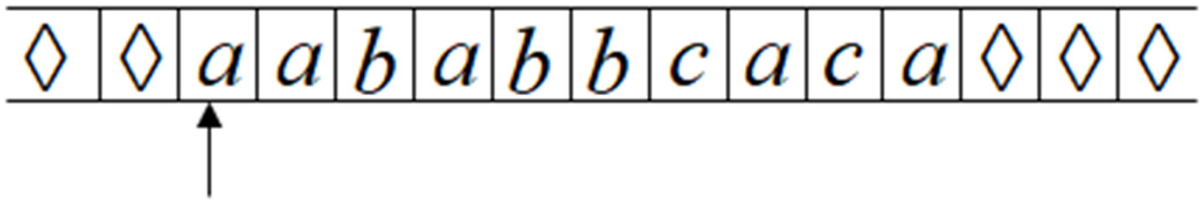
Simulation:

- **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

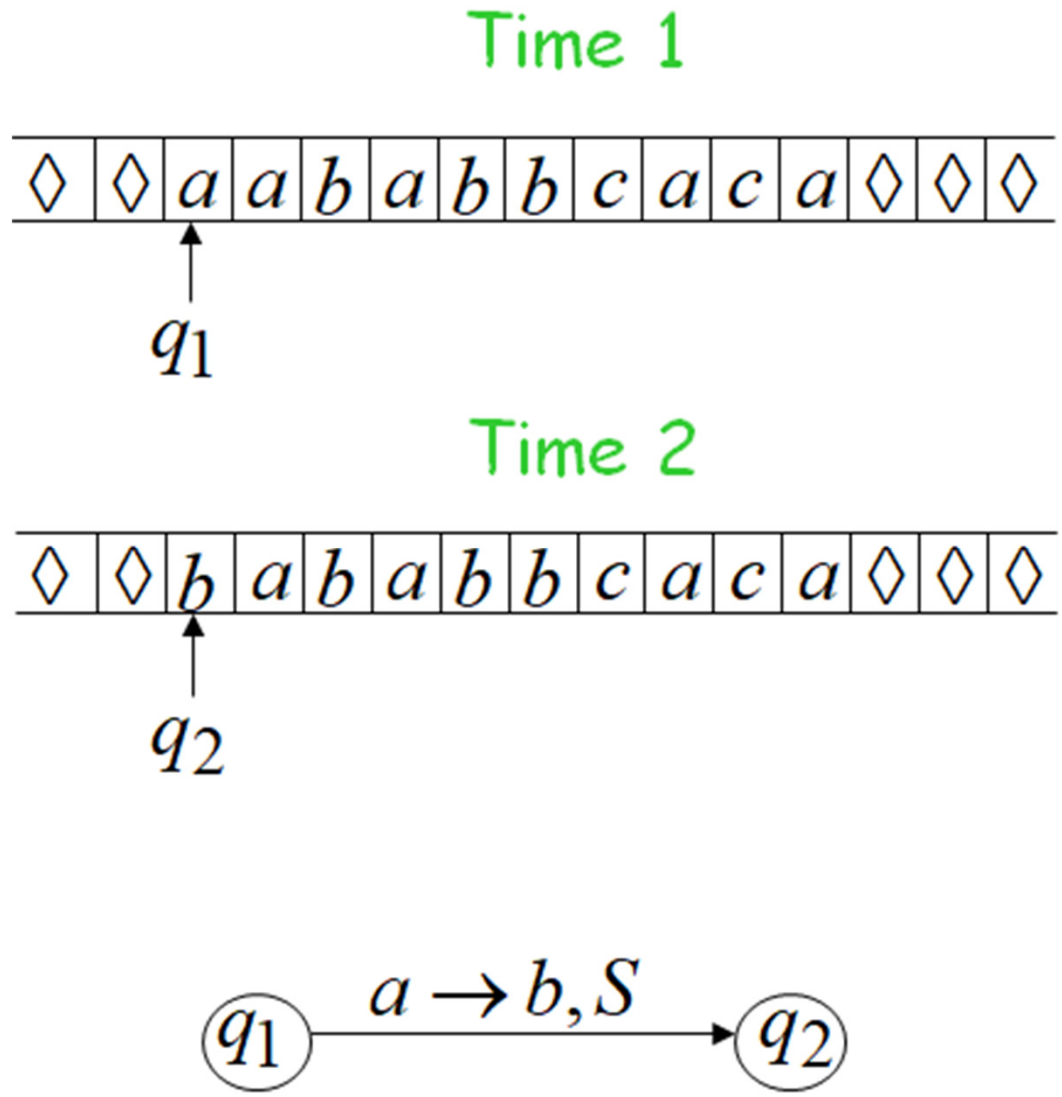
- Suppose in addition moving Left or Right, we give the option to the TM to **stay (S)** on the current cell (The head can stay in the same position), that is:

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



Moves: Left, Right, Stay

Turing Machines with Stay Option

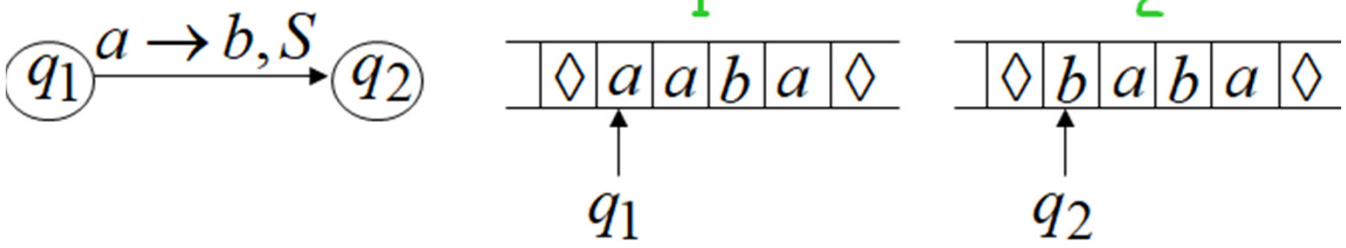


Turing Machines with Stay Option

- **Ordinary TMs and TMs with stay option have same power, and both of them accept Turing-recognizable languages.**
- **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.

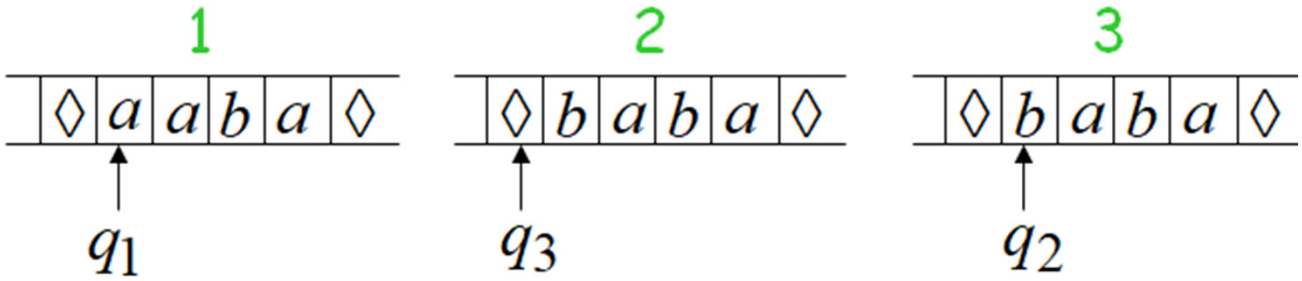
Turing Machines with Stay Option

Stay Option TM



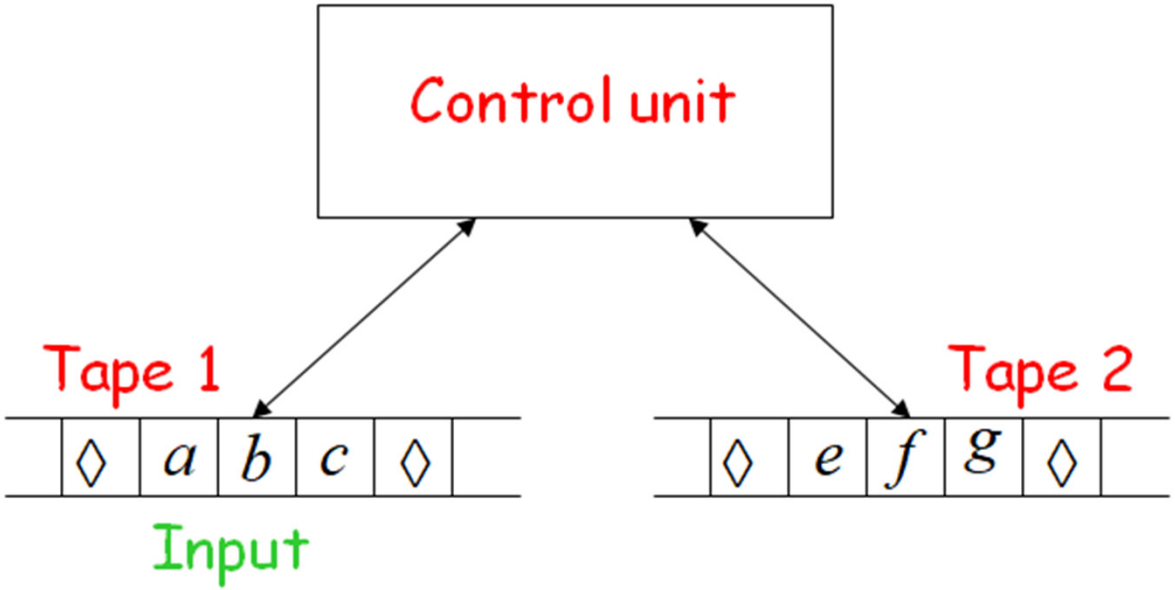
For every input x

Simulation in Standard TM



Multitape Turing Machines

- A **multitape Turing machine** is like an ordinary Turing machine with several tapes.
  - There are k tapes
  - Each tape has its own head for reading and writing.
  - Each tape head moves independently of the other heads
  - Initially the input appears on tape 1, and the others start out blank.



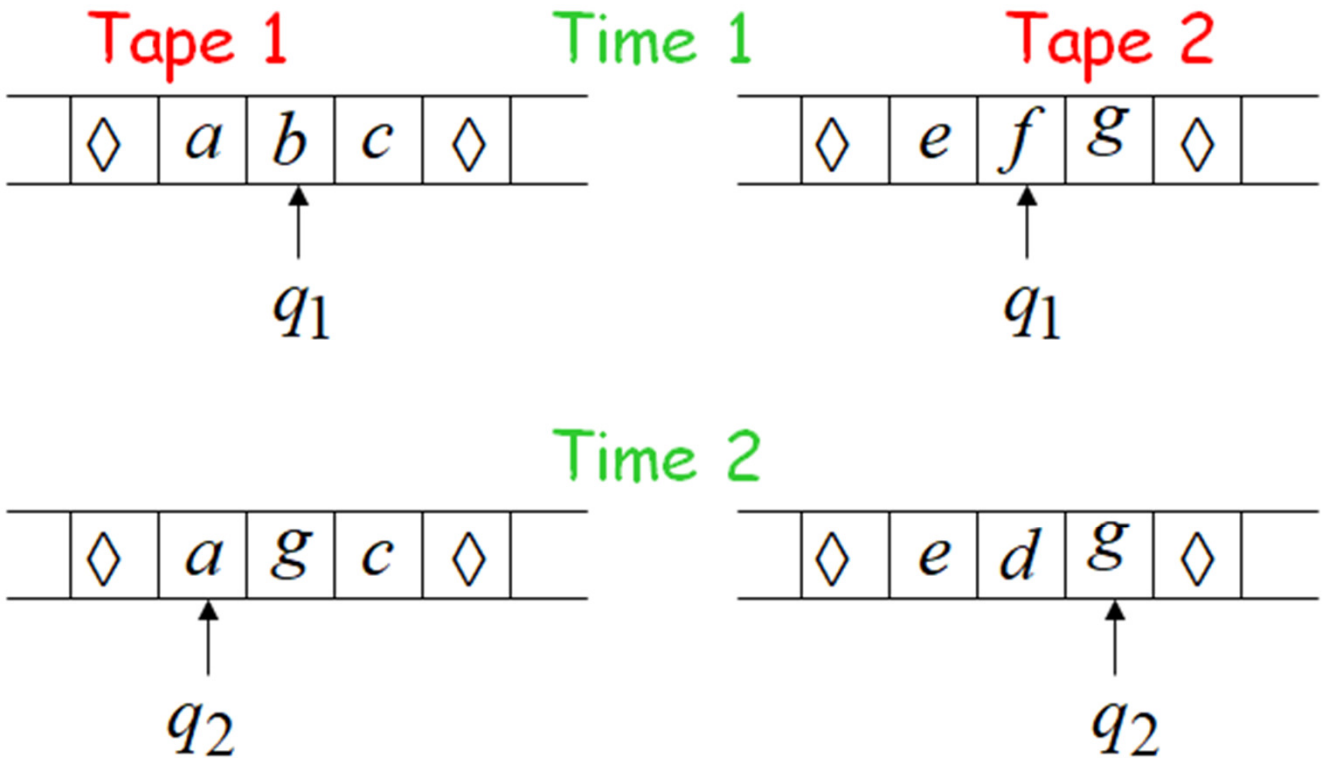
Multitape Turing Machines

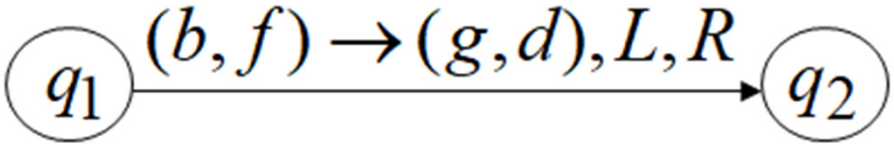
- The only fundamental difference from the ordinary TM is the state transition function.

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

- 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.

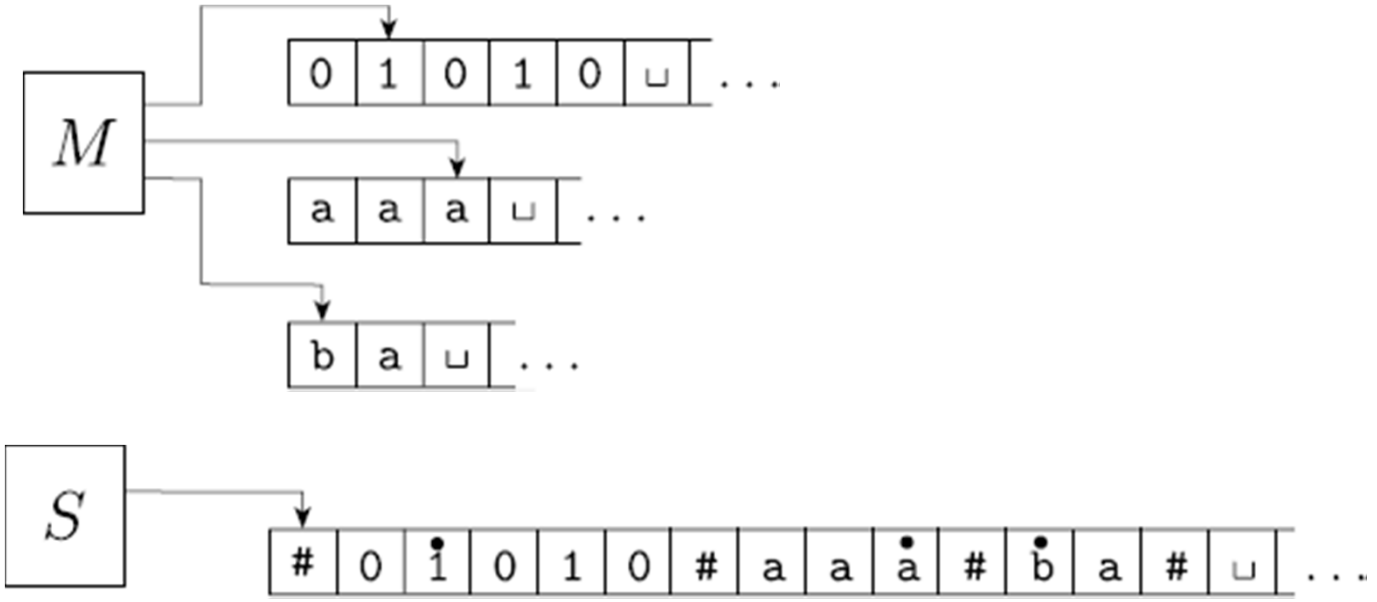
Multitape Turing Machines



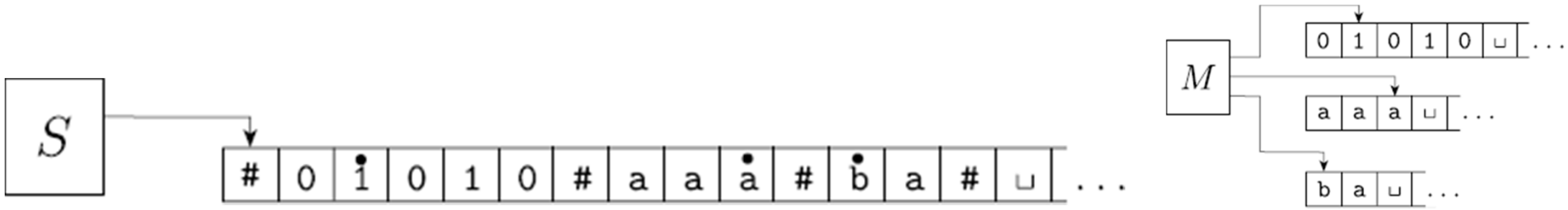


Multitape Turing Machines

- Multitape Turing machines appear to be more powerful than ordinary Turing machines, but we can show that they are equivalent in power, **and both of them accept Turing-recognizable languages.**
- **Every multitape Turing machine has an equivalent single-tape Turing machine.**
  - **show how to convert a multitape TM *M* to an equivalent single-tape TM *S*.**
    - ***Key Idea*: show how to simulate *M* with *S*.**



Simulating Multitape TM with Ordinary TM

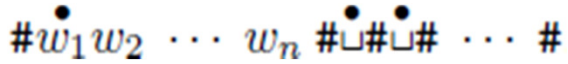


- We use # as a delimiter to separate out the different tape contents.
- To keep track of the location of heads, we use additional symbols
  - Each symbol in tape alphabet Γ has a **"dotted"** version.
  - A dotted symbol indicates that the head is on that symbol.
  - Between any two #'s there is only 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.



- 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 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".

Multitape Turing Machines

- Same power doesn't imply same speed.

Language $L = \{a^n b^n\}$

	Acceptance Time
Standard TM	n^2
Go back and forth	
Two-tape TM	n
Copy b^n to tape 2	
Leave a^n on tape 1	
Compare tape 1 and tape 2	

Nondeterministic Turing Machines

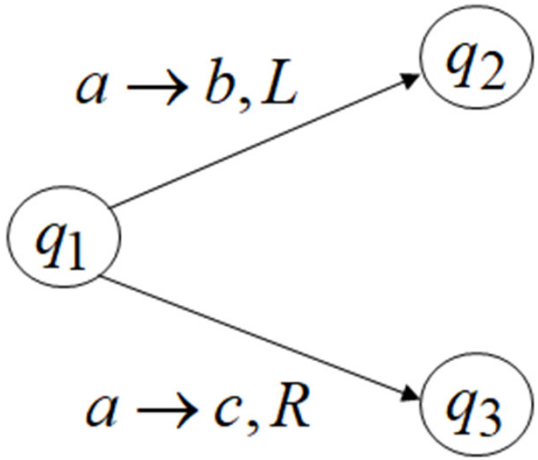
- **An ordinary TM is a deterministic machine.**
- The transition function of an **ordinary TM** is:

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

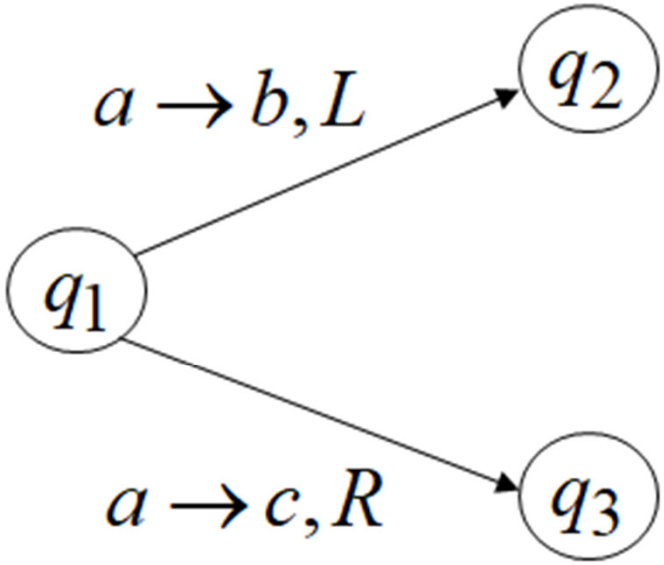
- A **nondeterministic TM** will proceed computation with multiple next configurations.
- The **transition function for a nondeterministic Turing Machine** has the form:

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

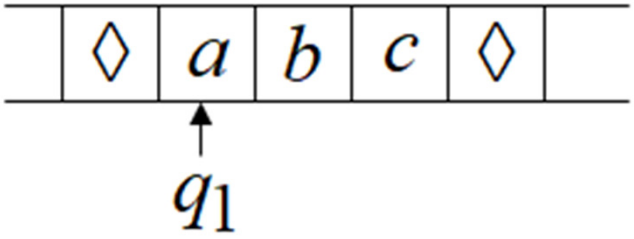
Non Deterministic Choice



Nondeterministic Turing Machines

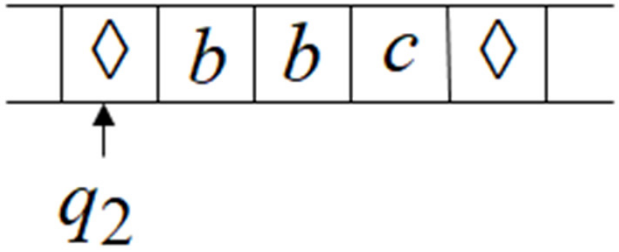


Time 0

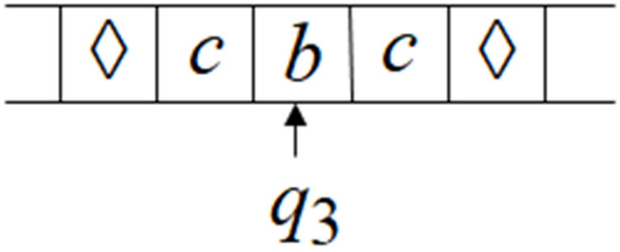


Time 1

Choice 1

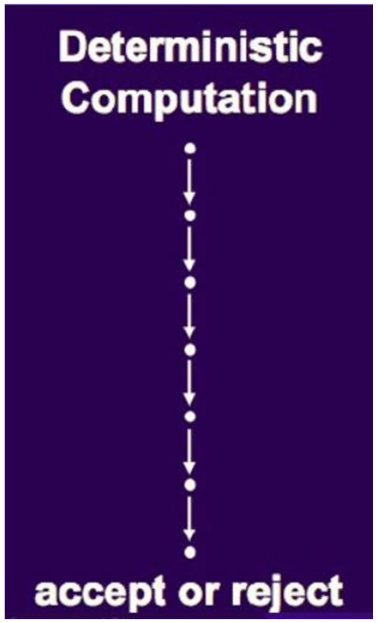


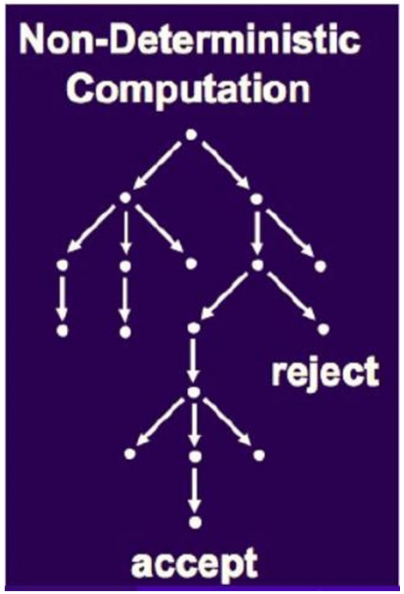
Choice 2



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.





Keep track of all possible computations

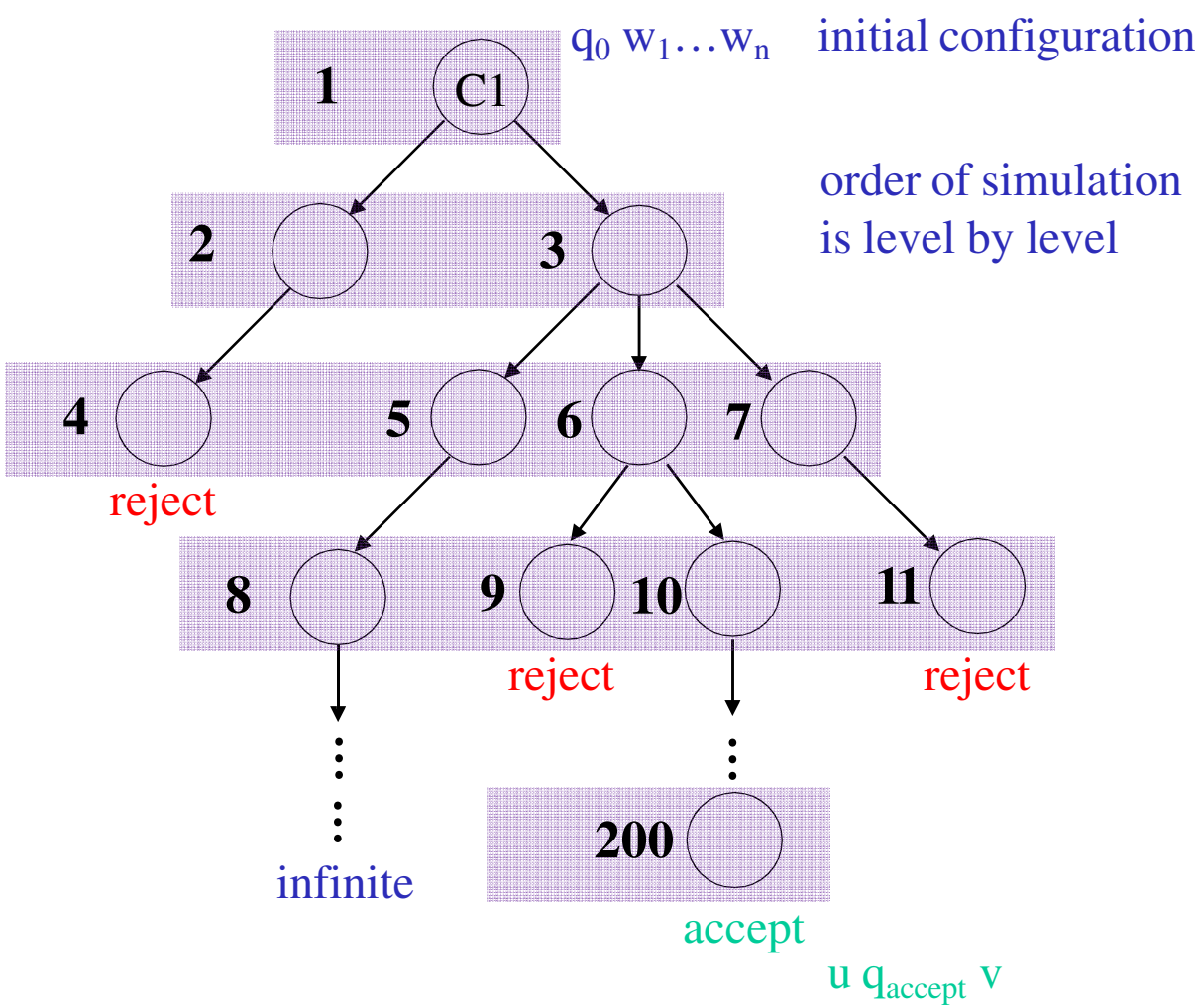
Nondeterministic Turing Machines

- What is the power of Nondeterministic TMs?
  - Is there a language that a nondeterministic TM can accept but deterministic TM can't accept? ➔ **NO**

- **Every nondeterministic Turing machine has an equivalent deterministic Turing Machine.**

- **We can simulate any nondeterministic TM *N* with a deterministic TM *D*.**
  - ***Key Idea:***
    - **D tries 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.**

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.

COM3064
Automata Theory

Week 11: Decidability and The class of P and NP

Lecturer: Dr. Sevgi YİĞİT SERT

Spring 2023

Resources: CS301 Formal Languages, Automata And Computatıon, Kemal Oflazer
CSE 135 Introduction to Theory of Computation, Sungjin Im
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli

Turing Machines and Decidability

- The most general model of computation
- Nondeterministic TMs are equivalent to Deterministic TMs.
- Multitape TMs are equivalent to Deterministic TMs.
- Computations of a TM are described by a sequence of configurations. (Accepting Configuration, Rejecting Configuration)

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be solved.

- Why discuss **unsolvability**?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

Recursive Languages

- A language L is recursive if there exists a Turing machine M which accepts all the strings in L and reject all the strings not in L.
- The Turing machine halts every time and gives an answer (accepted or rejected) for each and every string input.

For string : w

$w \in L$ then M halts in an accepting state

$w \notin L$ then M halts in a non-accepting state

- The set of palindromes over any alphabet is recursive.
- The set of prime numbers {2, 3, 5, 7, 11, 13, 17,…} is recursive.

Recursively Enumerable Languages

- A language L is recursively enumerable language if there exists a Turing machine M which accepts (and therefore halt) for all the strings in L.
- But may or may not halt for all input strings which are not in L.

For string : w

$w \in L$ then M halts in an accepting state

$w \notin L$ then M halts in a non-accepting state

or loops forever (goes on indefinitely)

- The term Turing-recognizable (or simply recognizable) languages is also used for recursively enumerable languages.

Decidable Languages

- A language L is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.
- There exists a TM M which decides L.
  - TM halts in an accepting configuration if w is in the language L.
  - TM halts in a rejecting configuration if w is not in the language L.

Undecidable Languages

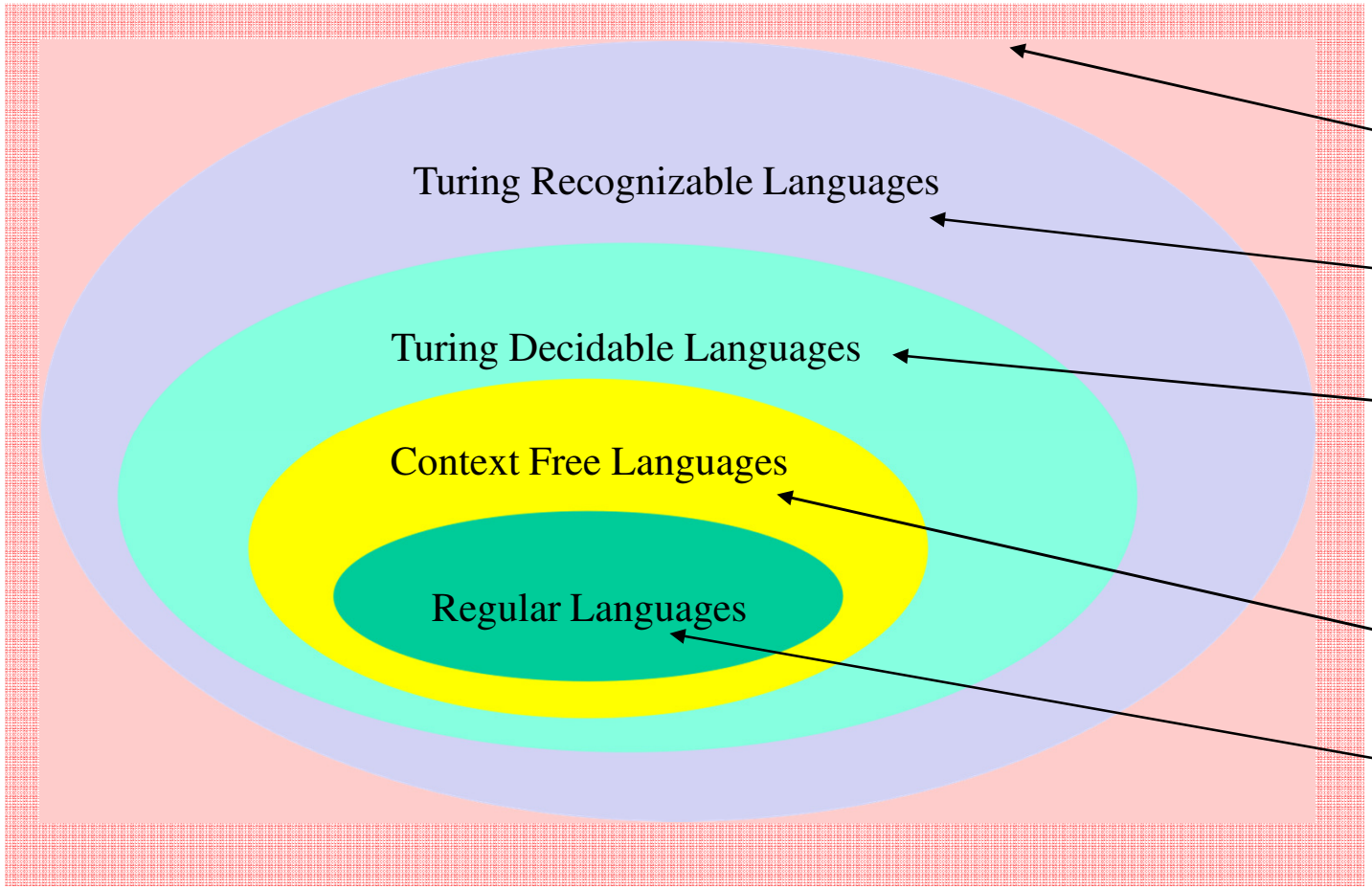
- A language L is undecidable if it is not decidable.
- There exists no Turing machine for that language.
- A famous undecidable problem is the Halting problem: the problem of determining whether a Turing machine halts (by accepting or rejecting) on a given input.

Input : Turing Machine M

String w

Question : Does M halt on input w ?

Decidability



Unsolvable Problems
- Turing-unrecognizable languages

Turing Machines

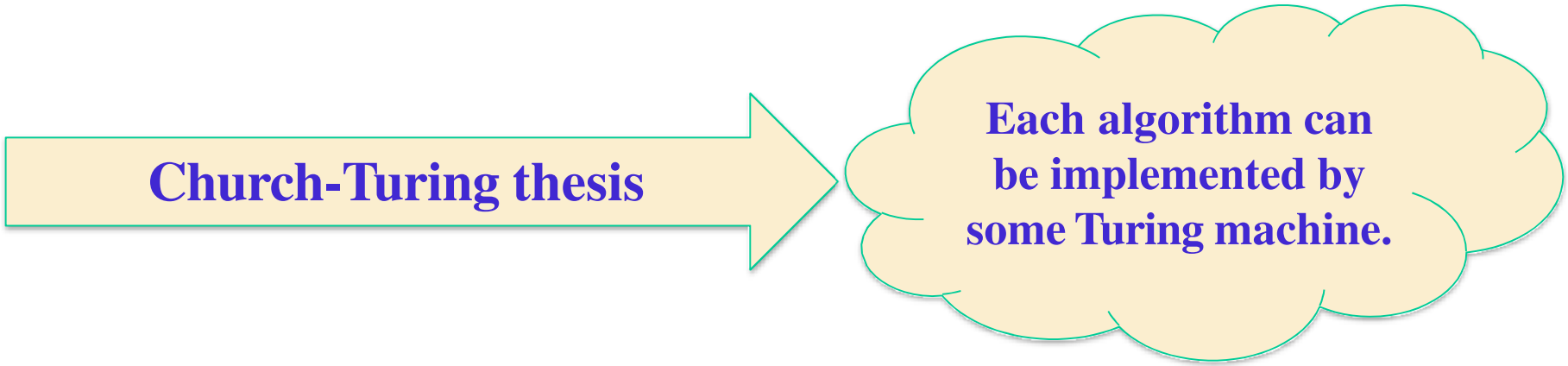
Turing Deciders

PDAs

DFAs

Church-Turing Thesis

- 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.
- An **algorithm** is a finite sequence of precise instructions for performing a computation or for solving a problem.



Church-Turing Thesis

- According to the Church Turing thesis, anything that is computable is computable by a Turing machine.
- A TM can do (compute) anything that a computer can do.
- Many other computation models have been proposed.
    - Recursive function (Gödel), λ-calculus (Church), formal language (Post), assembly language-like RAM (Shepherdson & Sturgis), boolean circuits (Shannon), extensions of the Turing machine (more strings, two-dimensional strings, and so on), etc.
- All have been proved to be equivalent.

Complexity Theory

- Assume that a problem (language) is decidable. Does that mean we can realistically solve it?

- NO, not always. It can require much of time or memory resources.

- Complexity Theory aims to make general conclusions of the resource requirements of decidable problems (languages).

- We only consider decidable languages and deciders.
  - Our computational model is a Turing Machine.
    - Time: the number of computation steps a TM machine makes to decide on an input of size n.
    - Space: the maximum number of tape cells a TM machine takes to decide on a input of size n.

Time Complexity

- How much time (or how many steps) does a single tape TM take to decide

$$A = \{0^k 1^k \mid k \geq 0\}?$$

- M = "On input w:
  1. Scan the tape and reject if w is not of the form 0^*1^*.
  2. Repeat if both 0s and 1s remain on the tape.
  3. Scan across the tape crossing off one 0 and one 1.
  4. If all 0's are crossed and some 1's left, or all 1's crossed and some 0's left, then reject; else accept

- How many steps does M take on an input w of length n?
  - The number of steps of M takes is n^2

Time Complexity

- The number of steps in measured as a function of n - the size of the string representing the input.

- In worst-case analysis, we consider the longest running time of all inputs of length n.
- In average-case analysis, we consider the average of the running times of all inputs of length n.

- Let M be a deterministic TM that halts on all inputs. The **time complexity of M** if the function $f : N \rightarrow N$, where $f(n)$ is the maximum number of steps that M uses on any input of length n. If $f(n)$ is the running time of M we say
  - M runs in time $f(n)$
  - M is an $f(n)$-time TM

The class P

- P is the class of languages that are **decidable in polynomial time** on a deterministic single-tape TM

$$P = \bigcup_k Time(n^k)$$

- The class P is important for two main reasons:
  1. P is **robust**: The class remains invariant for all models of computation that are polynomially equivalent to deterministic single-tape TMs.
  2. P roughly corresponds to the class of problems that are realistically **solvable on a computer.**

- A language is in P if and only if one can write a pseudo-code that decides the language in polynomial time in the input length; the code must terminate for any input

The class P - Example

PATH = $\{< G, s, t > \mid G$ is a directed graph with n nodes that has a path from s to $t\} \in$ P.

- M = "On input $< G, s, t >$

  1. Place a mark on s.
  2. Repeat 3 until no new nodes are marked
  3. Scan edges of G. If (a, b) is an edge and a is marked and b is unmarked, mark b.
  4. If t is marked, then accept else reject.

- Steps 1 and 4 are executed one and each takes at most $O(n)$ time on a TM.
- Steps 3 is executed at most n times and each execution takes at most $O(n^2)$ steps (∞ number of edges)

- Total execution time is a polynomial in n

The class NP

- For some problems, even though there is a exponentially large search space of solutions (e.g., for the path problem), we can avoid a brute force solution and get a polynomial-time algorithm.
- For some problems, it is not possible to avoid a brute force solution and such problems have so far resisted a polynomial time solution.

- HAM PATH:
  - A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once

- We are not aware of any algorithm that solves HAMPATH in polynomial time. But we know a brute-force algorithm finds a s-t Hamiltonian path in exponential time. Also we can verify/check if a given path is a s-t Hamiltonian path or not.

The class NP

- NP is the class of languages that have **polynomial time verifiers.**
- NP stands for **nondeterministic polynomial time**
- Problems in NP are called **NP-Problems.**