جامعة الملك سعود
King Saud University

Theory of Computation
CSC 339 – Spring 2021

Chapter-3: part2
Turing Machines

King Saud University

Department of Computer Science

Dr. Azzam Alsudais

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**
- **When we make statements, we need to ensure that they’re complete in the sense that a counterexample cannot be made.**

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**
- **When we make statements, we need to ensure that they're complete in the sense that a counterexample cannot be made.**
  - **e.g., $L = \{0^n1^n \mid n \geq 0\}$**

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**
- **When we make statements, we need to ensure that they're complete in the sense that a counterexample cannot be made.**
  - **e.g., $L = \{0^n1^n \mid n \geq 0\}$**
  - **Incomplete description of L may say something like: *L* consists of strings of equal number of 0's and 1's.**

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**

- **When we make statements, we need to ensure that they're complete in the sense that a counterexample cannot be made.**

  - **e.g., $L = \{0^n1^n \mid n \geq 0\}$**
  - **Incomplete description of L may say something like: *L* consists of strings of equal number of 0's and 1's.**

Using this description, then, the string 010101 should belong to ***L***. While, it is not!

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**

- **When we make statements, we need to ensure that they're complete in the sense that a counterexample cannot be made.**

  - **e.g., $L = \{0^n1^n \mid n \geq 0\}$**
  - **Incomplete description of L may say something like: *L* consists of strings of equal number of 0's and 1's.**

Using this description, then, the string 010101 should belong to ***L***. While, it is not!

It also doesn't say whether the empty string is part of ***L***.

Note on Completeness

- **Completeness is essential when it comes to theoretical studies.**

- **When we make statements, we need to ensure that they're complete in the sense that a counterexample cannot be made.**

  - **e.g., $L = \{0^n 1^n \mid n \geq 0\}$**
  - **Incomplete description of L may say something like: *L* consists of strings of equal number of 0's and 1's.**
  - **A more complete description, however, should say something like: *L* consists of strings that have a number of 0's *followed by* the same number of 1's, and L includes the empty string.**

Turing-decidable vs. Turing-recognizable

- **A language *L* is *Turing-recognizable* if and only if there is a Turing machine (TM) that accepts its strings.**

Turing-decidable vs. Turing-recognizable

- **A language *L* is *<u>Turing-recognizable</u>* if and only if there is a Turing machine (TM) that accepts its strings.**
  - **And for strings that ∉ L, the TM either <u>rejects</u> them or <u>loops forever</u>.**

Turing-decidable vs. Turing-recognizable

- **A language *L* is *Turing-recognizable* if and only if there is a Turing machine (TM) that accepts its strings.**
  - **And for strings that $\notin$ L, the TM either rejects them or loops forever.**
- **On the other hand, a language *L* is said to be *Turing-decidable* if and only if there is a TM that recognizes L, and always halts.**

Turing-decidable vs. Turing-recognizable

- **A language *L* is <u>*Turing-recognizable*</u> if and only if there is a Turing machine (TM) that accepts its strings.**
  - **And for strings that ∉ L, the TM either <u>rejects</u> them or <u>loops forever</u>.**
- **On the other hand, a language *L* is said to be <u>*Turing-decidable*</u> if and only if there is a TM that recognizes L, and always halts.**
  - **For strings ∈ L, the TM always accepts.**
  - **For strings ∉ L, the TM always rejects.**

Variants of Turing Machines

- **Turing machines have a number of variants that are equivalent in power with the original model.**
  - **TM with an additional tape head action (S: stay)**
  - **Multi-tape TM**
  - **Nondeterministic TM**
  - **Enumerators**

Stay-put Head Action

- **One variant of a TM is to augment the transition function with the ability to let the head stay put (denoted by *S*).**

Stay-put Head Action

- **One variant of a TM is to augment the transition function with the ability to let the head stay put (denoted by *S*).**

- **Behind the scenes, this is possible by using the original TM model and moving the head right and then back left.**

Stay-put Head Action

- **One variant of a TM is to augment the transition function with the ability to let the head stay put (denoted by *S*).**

- **Behind the scenes, this is possible by using the original TM model and moving the head right and then back left.**

$$a \rightarrow x, S$$

Stay-put Head Action

- **One variant of a TM is to augment the transition function with the ability to let the head stay put (denoted by *S*).**

- **Behind the scenes, this is possible by using the original TM model and moving the head right and then back left.**

$$a \rightarrow x, S$$

$$\Downarrow$$

$$a \rightarrow x, R$$

$$y \rightarrow L \qquad \text{for some } y \in \Gamma$$

Multi-tape Turing Machines

- **Each tape has an independent head for reading and writing.**

Multi-tape Turing Machines

- Each tape has an independent head for reading and writing.
- Initially, the input appears on tape 1, whereas other tapes start out blank.

Multi-tape Turing Machines

- **Each tape has an independent head for reading and writing.**
- **Initially, the input appears on tape 1, whereas other tapes start out blank.**
- **The transition function is modified to allow for reading, writing, and moving the heads on some or all tapes simultaneously.**

Multi-tape Turing Machines

- **Each tape has an independent head for reading and writing.**
- **Initially, the input appears on tape 1, whereas other tapes start out blank.**
- **The transition function is modified to allow for reading, writing, and moving the heads on some or all tapes simultaneously.**

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

where k is the number of tapes

Multi-tape Turing Machines

$$\delta(q_i, a_1 , \ldots, a_k) = (q_j , b_1 , . . . , b_k , L, R, . . . , L)$$

If the machine is in state q_i

Multi-tape Turing Machines

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

If the machine is in state q_i and reads symbols a_1 through a_k from tapes 1 through k.

Multi-tape Turing Machines

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

If the machine is in state q_i and reads symbols a_1 through a_k from tapes 1 through k.
Then, the machine goes to state q_j

Multi-tape Turing Machines

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

If the machine is in state q_i and reads symbols a_1 through a_k from tapes *1* through *k*. Then, the machine goes to state q_j, writes symbols b_1 through b_k

Multi-tape Turing Machines

$$\delta(q_i, a_1 , \ldots, a_k) = (q_j , b_1 , \ldots , b_k , L, R, \ldots , L)$$

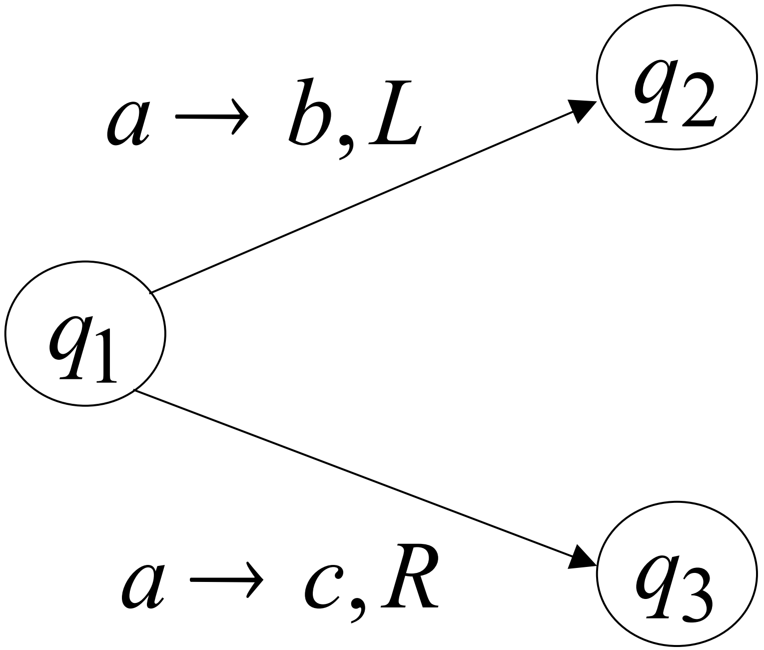
If the machine is in state q_i and reads symbols a_1 through a_k from tapes *1* through k. Then, the machine goes to state q_j, writes symbols b_1 through b_k, and directs each tape head to move *left*, *right*, or *stay put*.

Nondeterministic Turing Machines

- **Just like NFA, nondeterministic Turing machines can traverse multiple "paths" at the same time.**

Nondeterministic Turing Machines

- Just like NFA, nondeterministic Turing machines can traverse multiple “paths” at the same time.



Variants of Turing Machines

- **Along with the original model of Turing machines, these variants of TM planted the seed for modern computing.**

Variants of Turing Machines

- **Along with the original model of Turing machines, these variants of TM planted the seed for modern computing.**

- **Numerous use-cases can be realized and implemented using those variants of TM.**

Variants of Turing Machines

- **Along with the original model of Turing machines, these variants of TM planted the seed for modern computing.**

- **Numerous use-cases can be realized and implemented using those variants of TM.**

- **The concept behind Turing machines is a great embodiment of computability theory.**

Algorithms

- **What's an algorithm?**

Algorithms

- **What’s an algorithm?**
  - **A set of clearly-described ordered instructions that carry out a certain task.**

Algorithms

- What's an algorithm?
  - A set of clearly-described ordered instructions that carry out a certain task.
- Turing machines capture all algorithms: we can design a TM that simulates a given algorithm.

Algorithms - Example

- **Finding the integral root of a polynomial.**
  - $4x^3 - 2x^2 + x - 7$

Algorithms - Example

- **Finding the integral root of a polynomial.**
  - **$4x^3 - 2x^2 + x - 7$**
- **The integral root is an assignment of an integer value to the variables of the polynomial.**

Algorithms - Example

- Finding the integral root of a polynomial.
  - $4x^3 - 2x^2 + x - 7$
- The integral root is an assignment of an integer value to the variables of the polynomial.
- Can we design an algorithm that tells us whether a given polynomial has an integral root?

Algorithms - Example

- **Finding the integral root of a polynomial.**
  - $4x^3 - 2x^2 + x - 7$
- **The integral root is an assignment of an integer value to the variables of the polynomial.**
- **Can we design an algorithm that tells us whether a given polynomial has an integral root?**
  - **Short answer: NO.**

Algorithms - Example

- **However, we can design an algorithm (or a TM) that finds an integral root of a polynomial.**

Algorithms - Example

- However, we can design an algorithm (or a TM) that finds an integral root of a polynomial.

- But, if there isn't an integral root, then the algorithm would run forever.

Algorithms - Example

- However, we can design an algorithm (or a TM) that finds an integral root of a polynomial.

- But, if there isn't an integral root, then the algorithm would run forever.

- In this case, the TM is a <u>recognizer</u>, but not a <u>decider</u>.

Algorithms - Example

- **However, we can design an algorithm (or a TM) that finds an integral root of a polynomial.**

- **But, if there isn't an integral root, then the algorithm would run forever.**

- **In this case, the TM is a <u>recognizer</u>, but not a <u>decider</u>.**

- **Though, we can convert the TM to be a decider if we provide <u>bounds</u> within which the integral root should fall within.**

Algorithms - Example

- **However, we can design an algorithm (or a TM) that finds an integral root of a polynomial.**

- **But, if there isn't an integral root, then the algorithm would run forever.**

- **In this case, the TM is a <u>recognizer</u>, but not a <u>decider</u>.**

- **Though, we can convert the TM to be a decider if we provide <u>bounds</u> within which the integral root should fall within.**

- **For instance, the integral root should be within the range (x, y)**

Homework

- **Read page 193 and section 4.1**