Computation theory

Lecture 1

Brwa R. Hassan

Theory of Computation

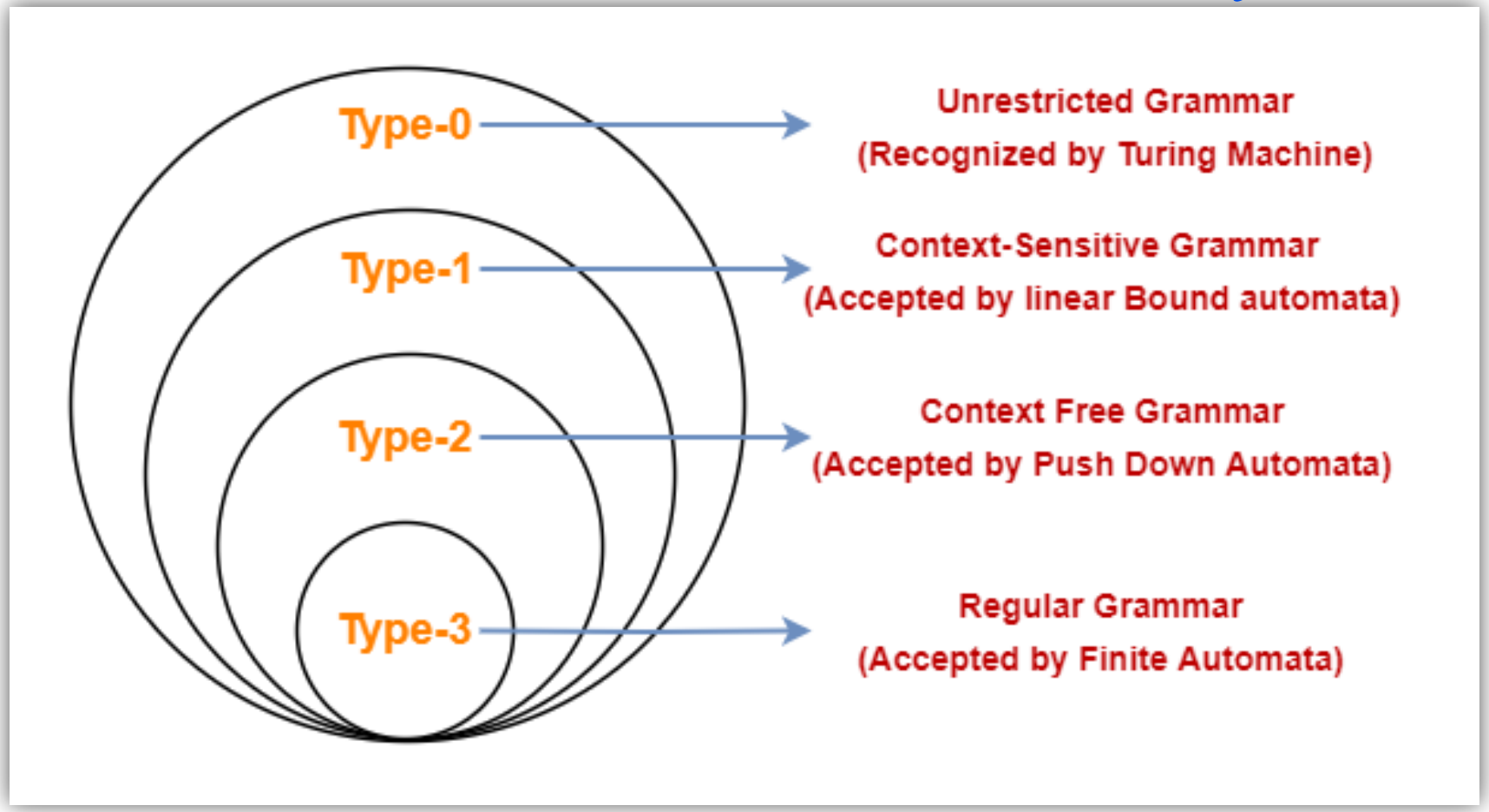
- Theory of computation is a part of theoretical Computer Science, mainly concerned with the study of how problems can be solved using algorithms

- Computation is calculation, solving, making decision or any task done by computer/calculator/ any machine.

- The term Theory defines capabilities, limitations of those machines.

- Purpose of the Theory of Computation: It attempts to find deep understanding of computational processes through the use of mathematical tools and models

hierarchy of computational models



hierarchy of computational models Definitions

- **Type 0: Unrestricted Grammar** : Language recognized by Turing Machine is known as Type 0 Grammar. They are also known as Recursively Enumerable Languages.
- **Type 1: Context-Sensitive Grammar :** Languages recognized by Linear Bound Automata are known as **Type 1 Grammar**. Context-sensitive grammar represents context-sensitive languages.
- **Type 2: Context-Free Grammar :** Languages recognized by Pushdown Automata are known as **Type 2 Grammar**. Context-free grammar represents context-free languages.
- **Type 3: Regular Grammar :** Languages recognized by Finite Automata are known as Type 3 Grammar. Regular grammar represents regular languages.

Key characteristics of regular language

- **Recognizable by a finite automaton**: A regular language is any language for which a deterministic or non-deterministic finite automaton can be designed. This is a state-based machine with a finite number of states and transitions.
- **Describable by a regular expression**: They can be formally defined using regular expressions, a sequence of characters that specify a search pattern.
- **Generated by regular grammars**: They are the set of strings produced by a regular grammar, which are simple production rules for generating strings.
- **Closed under certain operations**: Regular languages are "closed" under operations like union, concatenation, and Kleene star. This means if you perform these operations on regular languages, the result is still a regular language.
- **Do not support arbitrary nesting**: They cannot handle arbitrary levels of nesting or recursion, which requires more complex language classes. For example, the language of balanced parentheses is not a regular language.

Examples:

- The set of all strings over the alphabet {a, b} is a regular language.
- The language of all strings with any number of 'a's followed by any number of 'b's is a regular language.
- Any finite language is a regular language.

FSM (Finite State Machine)

- This is the simplest computational model. An FSM has a limited "memory" and can only process "regular languages," which are the simplest type of languages.

- It works by moving from one state to another based on input, without any ability to "look back" or remember past inputs beyond its current state. FSMs are useful for tasks with predictable patterns, like validating simple sequences of characters.

CFL (Context-Free Language)

• A more powerful model, represented by Pushdown Automata (PDA), which have a stack for memory. This allows them to recognize context-free languages (CFL), which include balanced structures like parentheses in expressions.

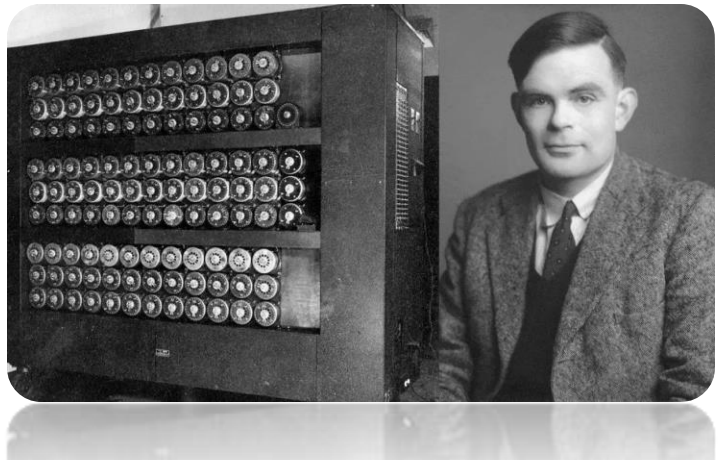
• With a stack, PDAs can handle patterns with nesting, such as {(a^n)(b^n)}, where n represents the number of matching pairs. However, PDAs still have limitations and can't solve more complex language structures.

Turing Machine

• Turing Machines represent a universal computational model. They have an infinite tape (memory) that can read and write, allowing them to compute any problem that is "algorithmically solvable.“

• Turing Machines can recognize recursively enumerable languages, which include

all computable problems. They are the foundation for modern computers and programming, as they can simulate any algorithm



Undecidable

- This category represents problems that cannot be solved by any algorithm, even by a Turing Machine. These are "undecidable" problems, meaning there's no way to create a procedure that can provide an answer for all cases.

- Examples include the famous Halting Problem, where we can't determine if an arbitrary program will finish running or loop forever. These problems show the theoretical limits of computation.

Compiler Design

Finite automata perform lexical analysis to tokenize source code, identifying keywords, identifiers, and operators during compilation.

Cryptography

Computational complexity underpins secure algorithms, digital signatures, and blockchain technologies by leveraging hard-to-solve problems.

AI & Machine Learning

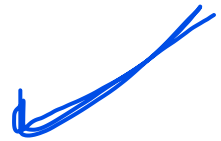
Theoretical models enable algorithm design for learning, natural language processing, and pattern recognition in intelligent systems.

Software Verification

Formal methods use automata and logic to mathematically prove correctness, safety, and reliability of critical software systems.

Applications of Theory of Computation in Computer Science

Theory of Computation



Theory of computation is divided into three major branches:

1. **Automata Theory and language** (deals with the mathematical model of computation)

2. **Computability Theory** (Is a certain problem solvable?)

3. **Computational Complexity Theory** [Efficiency] (How hard a certain problem is?)

Automata theory

• Deals with the definitions and the properties of mathematical models of computation.

• One of these models is called **Finite Automata** which is used in **text processing, Compiler and hardware design.**

• Another model is called Context Free Grammar that's used in programming and AI .

• The following are some terminologies related to Automata Theory.

Languages

• Till half of this century several people define the language as a way of understanding between the same group of beings, between human beings, animals, and even the tiny beings, this definition includes all kinds of understanding, talking, special signals and voices.

1. Talking language : (e.g.: English, Arabic)
2. 2. Programming language: (e.g.: c++, Pascal)
3. 3. Formal language: (any language we want.)

Languages

• This definition works till the mathematician called Chomsky said that: the language can be defined mathematically as:

1. A set of letters which called Alphabet. This can be seen in any natural language, for example the alphabet of English can be defined as:

• E = {a, b,…, z}

2. By concatenate letters from alphabet, we get words.

3. All words from the alphabet make language.

Mathematical Terminology:

- **Symbol**: is the basic building block of ToC. Example: Can be anything like: a,b,c,A,B,Z,0,1,…etc
- **Alphabet**: is a finite set of symbols. We use the symbol ∑ (sigma) to denote an alphabet. Examples:
  - ∑ = {0,1}: Binary alphabet
  - ∑ = {A~Z, 0~9}: Alphanumeric alphabet
  - ∑ = {a,b,c, ….,z}: Alphabet of small letters

Mathematical Terminology:

- A **string** or word is a finite sequence/group of symbols chosen from the alphabet (∑)
- Examples: 01011 = is a string from the binary alphabet ∑{0,1} abacbc = is a string from the alphabet ∑{a, b, c} 3786 = is a string over {0,1,2,3,4,5,6,7,8,9}
- **Empty string** is the string with no symbols, denoted by ε (epsilon) or λ (lambda)
- **Length of a string**: denoted by |w|, is equal to the number of symbols/characters in the string.

Mathematical Terminology:

- Example:

w = classroom |w| = 9

w = 010100 |w| = 6

w = ε |w| = 0

- The position of a symbol in a string is denoted by (w)

Example: w = classroom w(3) = a, w(4) = s, w(5) = s

- **Concatenation of strings**: x = abc, y = pqr
- Concatenation of x and y, x◦y (or xy) = abcpqr

Mathematical Terminology:

Power of Alphabet: Power of Alphabet: If ∑ is an alphabet, then, ∑k = is the set of all strings of length k. Example: Let, ∑ = {0,1}, then

$$\Sigma^0 = \{\lambda\}$$
$$\Sigma^1 = \{0,1\}$$
$$\Sigma^2 = \{00,01,10,11\}$$
$$\Sigma^3 = \Sigma^2\Sigma$$
$$= \{00,01,10,11\}\{0,1\} = \ldots$$
$$= \{000,001,010,011,100,101,110,111\}$$

Mathematical Terminology:

- **Language**: is a set of strings chosen from the alphabet ∑ Language L could be finite or infinite
- Example: Suppose ∑ = { a,b }
- L1 = Set of all strings of length 2= { aa, ab, ba, bb } [Finite]
- L2 = Set of all strings of length 3 = { aaa, aab, aba, abb, baa, bab, bba, bbb }[Finite]
- L3 = Set of all strings where each string starts with “a” = { a, aa, ab, aaa, aab, aba, abb,…….. } [Infinite]

Mathematical Terminology:

- Example:
- **Alphabetic**: ∑= {0, 1}.
- **Sentences**: 0000001, 1010101.
- **Rules**: Accept any sentence start with zero and refuse sentences that start with one
- So we accept: 0000001 as a sentence satisfies the rules.
- And refuse: 1010101 as a sentence doesn't satisfy the rules.

Mathematical Terminology:

- Example:
- **Alphabetic**: ∑= {a, b}.
- **Sentences**: ababaabb, bababbabb
- **Rules**: Accept any sentence start with a and refuse sentences that start with b.
- So we accept: aaaaabba as a sentence satisfies the rules._
- And refuse: baabbaab as a sentence doesn't satisfy the rules.

Some Rules in Mathematic

Commutative Law

The order of sets does not matter for union or intersection.

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative Law

How sets are grouped does not change the result of multiple unions or intersections.

$$(A \cup B) \cup C = A \cup (B \cup C)$$

$$(A \cap B) \cap C = A \cap (B \cap C)$$

Distributive Law

Connects union and intersection operations.

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

De Morgan's Laws

Describe the relationship between the complement of a union/intersection and the intersection/union of the complements.

$$(A \cup B)' = A' \cap B'$$

$$(A \cap B)' = A' \cup B'$$

Identity Law

The union of a set with the universal set U is U, and the intersection with the empty set $\varnothing$ is $\varnothing$.

$$A \cup U = U$$

$$A \cap \varnothing = \varnothing$$

Idempotent Law

The union or intersection of a set with itself is the set itself.

$$A \cup A = A$$

$$A \cap A = A$$

Some operations

- **Function reverse**: if a is a word in some language, then reverse of a is the same string of letter spelled backward.

- **Example**: lets a=xxxx, b=543, c= aab then reverse(a)=xxxx, reverse(b)=345, reverse(c)=baa.

- Here we want to mention that if we apply this function on words some times the result does not satisfy with the definition of the language.

- Example Let A an alphabet of the language L1 be {0 1 2 3 4 5 6 7 8 9} Let L1 = {all words that does not start with zero} And c=210 Then reverse(c)=012, which is not in L1.

Some operations

- **Concatenation**: combines two strings by putting them one after the other.
- Example: x = united, y = states, then xy = unitedstates, or simply yx = statesunited
- The concatenation of the empty string with any other string gives the string itself:

$x \lambda = \lambda x = x$

- Substring: String x is a Substring of y if x appears in y as a single element or a serious of related elements.
- Example: for y=abcde , x= bcd. x is a substring of y but w=ac is not a substring of y

Some operations

- **Prefix & Suffix**: a string x is a prefix of string y if x appears as a substring in the beginning of y. correspondingly, if x appears in the end of y then x is a Suffix of y.
- Example: for y= abcde then we have abc as a prefix of y, and de as a Suffix of y.

Some operations

- **Concatenation** : If L1 and L2 are languages, then L = L1. L2 (or simply L1L2) is the set:
- **Union** L = {xy , x ϵ L1, y ϵ L2} L1∪ L2 = {x| x ϵ L1 or x ϵ L2}
- **Kleene star** of a language L is the set of all strings obtained by concatenating zero or more strings from L. It is denoted by L*. L*={x1x2x3….xk | k≥0 and xi ϵ L}

Some operations

- Let L1= {white, black} and L2 = {chocolate, cream} from the alphabet Σ of the 26 English letters, then
- L1∪L2 = {white, black, chocolate, cream}
- L1. L2 = {whitechocolate, whitecream, blackchocolate, blackcream}.
- L1*={λ, white, black, whitewhite, whiteblack,blackwhite, blackblack, whitewhitewhite, whitewhitewhietblack,…… }

Thank you