CS323: Theory of Computation

Lecture 1

Introductory Lecture

Dr. Samar Hesham

samar@fci-cu.edu.eg

Department of Computer Science

Faculty of Computers and Artificial intelligence

Cairo University

Egypt

Syllabus and Terminologies

- Regular Languages .. Regular Sets
  - REs (Regular Expressions)
  - FSMs (or FSA/FA) … Finite State Machines/Automata
    - DFA vs. NFA … Deterministic vs. Non-deterministic FSA
      - Comparison and conversion
    - Examples & Closure Operations
    - Pumping Lemma
- Context Free Languages
  - CFGs … Context Free Grammars
  - PDA … Push Down Automata
  - Parsing: CFG generating strings vs. PDA recognizing strings
- Turing Machine

Resources

Text Books:

- "***Introduction to Computer Theory***", 2nd Edition, by *Daniel I.A. Cohen, John Wiley & Sons*.

- "***An Introduction to Formal Languages & Automata***" by Peter Linz, Jones & Bartlett Publishers, 3rd edition, Inc. January 2001; ISBN: 0763714224.

- "***Theory of Computation an Introduction***" by James L. Hein, Jones & Bartlett Publishers 1996; ISBN: 0-86720-497-4.

Course Grading Policy

Midterm Exam	20
Lab Tasks & Practice	20
Final	60
Total	100

Why to study *Theory of Computation*?

- It is the most abstract course in the whole curriculum
- But a fundamental course for all CSians
- This course is not about how to develop a program or build a computer
- It helps you know a lot more about how people looked to CS *as a science* in the past 70 years

Why to study *Theory of Computation*?

- It is about:
  - What kind of things can we really compute?
  - How fast we can do it? and
  - How much space/memory it needs to be computed?

Why to study *Theory of Computation*?

- This course is about different models of computations you can use to compute stuff
- But ***abstracted*** from the hardware/machine specifications
- We will deal with kinds of problems that takes ***inputs***, and ***process it***, then ***decide*** whether these inputs are ***acceptable*** or not
- Inputs are set of symbols formed by an alphabet (binary, English alphabet, ...)

Why to study *Theory of Computation*?

- There are a lot of applications that connect to this course:
  - Theory behind writing *compilers/translators* for programming languages
  - *Computer Architecture* → model particular processes/states using finite-state-machines (FSMs)
  - *String search* algorithms, *word processing* algorithms, any kind of editors → that is a RE/FSM
  - When you do a representation of a language like XML, it is just a *grammar-writing*

Turing Machine

- Turing Machine invented by *Alan-Turing* in 1940's
- It is an abstraction of how programming languages and computers work nowadays
- He invented a *mathematical abstraction* of a complete representation of how might we can do a computation
- Any problem that can be computed by a program on a computer, it can be *computed by* (*simulated on*) a Turing Machine

Turing Machine

- Turing Machine is like a *human with organs*, cutting one organ (going down into lower machine) will make it handicap, or taking higher time and resources
- The lower level is the FSM, and then add up pieces to form more powerful machines (till getting a Turing Machine)
- Then you will get into the *twilight zone* that has problems can't be computed

Definition of a language "L"

∑ is the set of alphabets:

Language = set of strings can be formed by ∑

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Definition of a language "L"

- Language: is a **<u>set</u>** of strings or it *is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"

  - *If A is an alphabet, then a language over A is a collection of all strings over A.*

  - *A* is the biggest possible language over A, and every other language over A is a* ***<u>subset</u>*** *of A*.*

- ***<u>Example</u>***

  If A={a} then the following are simple examples of languages over an alphabet A:

  $\{\Phi\}$, $\{\lambda\}$, $\{a\}$, *and* $\mathbf{A^*} = \{\lambda, a, aa, aaa, \ldots\ldots\ldots\}$.

Note: λ, Λ, **ε** are used interchangeably in textbooks

Example

Let the alphabet be $\Sigma = \{a, b\}$ then,

- *$\{\Phi\}$ = Empty Language*
- *$\{\lambda\}$ = Empty Character*
- *$\{a\}$ = Letter 'a'*
- *$\{b\}$ = Letter 'b'*
- **a^*** $=\{\lambda, a, aa, aaa, \ldots\ldots\ldots\}$
- **$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, bbb, \ldots\}$**
- **$\Sigma^+ = \Sigma^* - \{\lambda\}$**
- **$L_1 = \{a, aa, bbb\}$ finite**
- **$L_2 = \{a^n b^n : n >= 0\}$ infinite**
- **$\overline{L_1} = \Sigma^* - L_1$**

Regular Languages/Sets … RLs/RSs

- It is a language that:
  - Can be governed/expressed by rules called ***Regular Expressions*** (REs).

Theorem

$$\left\{\begin{array}{l}\text{Languages}\\ \text{Generated by}\\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Properties of Regular Languages

For regular languages L_1 and L_2 we have

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

$\left.\begin{array}{l} \\ \\ \\ \end{array}\right\}$ Are regular Languages

Properties of Regular Languages

For regular languages L_1 and L_2 we have

$L_4 = \{a, bc, x, yz\}$

Union: $L_1 \cup L_2$

$L1 = \{a, bc\}$
$L2 = \{x, yz\}$

Concatenation: $L_1 L_2$

$L3 = \{ax, ayz, bcx, bcyz\}$

$\{\lambda, a, ab, aa, abab, aab, \ldots\}$

Star: $L_1{}^*$

$L1 = \{a, ab\}$

Regular Expressions

- RE is a **syntax** for describing simple/regular languages and patterns.
- REs are also used in applications that involve file parsing and text matching.
- Many implementations have been made for RE.

RE Rules

1. Let $\sum = \{x\}$, then $L = \sum^*$

In RE, we write x^*

2. Let $\sum = \{x, y\}$, then $L = \sum^*$

In RE, we write $(x+y)^*$

3. Kleene's star "*" means any combination of letters of length zero or more.

output strings

$* = 0$ λ

$* = 1$ $\{x\}$ or $\{y\}$

$* = 2$ $\{xx, xy, yx, yy\}$

$* = 3$ $\{xxx, xxy, \ldots\}$

RE Rules

Given an alphabet ∑, the set of regular expressions is defined by the following rules.

1. For every letter in ∑, the letter written in bold is a regular expression. **Λ-lamda or ε-epsilon** is a regular expression. Λ **or ε** means an empty letter.
2. If $\mathbf{r_1}$ and $\mathbf{r_2}$ are regular expressions, then so are:
    1. **($\mathbf{r_1}$)**
    2. $\mathbf{r_1\,r_2}$ **and**
    3. $\mathbf{r_1+r_2}$ **or**
    4. $\mathbf{r_1^*}$ **and also** $\mathbf{r_2^*}$

NOTE 1: r_1^+ is not a RE

NOTE 2: spaces are ignored as long as they are not included in ∑

3. Nothing else is a regular expression.

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad a$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1 * \\ (r_1) \end{array}\right\} \text{ Are regular expressions}$$

Examples

A regular expression: $(a+b\cdot c)^*\cdot(c+\varnothing)$

Not a regular expression: $(a+b+)$

Languages of Regular Expressions

- $L(r)$: language of regular expression r

$a^* = \{\lambda, a, aa, aaa, aaaa, \ldots\}$

- Example:
  - If $r = (a + b \cdot c)^*$

$= \{\lambda, a, bc, aa, abc, bcbc, bca$

$$L((a+b\cdot c)^*) = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

Regular Expressions

- Regular expressions
  - describe regular languages
- Example:

  Describe the language

$$(a+b\cdot c)^*$$

$$\{a,bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

Definition (continued)

- For regular expressions r_1 and r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

Example

- Regular expression: $(a+b)\cdot a^*$

$$\begin{aligned}
L(\underbrace{(a+b)}_{r_1}\cdot \underbrace{a^*}_{r_2}) &= L((a+b))\,L(a^*) \\
&= L(a+b)\,L(a^*) \\
&= (L(a)\cup L(b))\,(L(a))^* \\
&= (\{a\}\cup\{b\})\,(\{a\})^* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}$$

Example

- Regular expression $r = (aa)^*(bb)^*b$

- What is the regular languages that r describes?

$$L(r) = \{a^{2n}b^{2m}b: \quad n, m \geq 0\}$$

Equivalent Regular Expressions

- Definition:

Regular expressions r_1 and r_2

are **equivalent** if

$$L(r_1) = L(r_2)$$

Example

L = {all strings with no two consecutive 0's }

$$r_1 = (1 + 01)^*(0 + \lambda)$$

$$r_2 = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda)$$

$$L(r_1) = L(r_2) = L$$ $\Rightarrow$ r_1 and r_2 are equivalent regular expr.

Example

- Regular expression $r = (0+1)^*\,00\,(0+1)^*$

$L(r)$ = { all strings with at least
two consecutive 0's }

RE Examples

RE-1

- Example 1

$\sum = \{a, b\}$

- Formally describe all words with a followed by any number of b's

$L = a\ b^* = ab^*$

- Give examples for words in L

{a ab abb abbb …..}

RE-2

- Example

$\sum = \{a, b\}$

- Formally describe all words with a followed by one or more b's

$L = \cancel{a\,b}^{+} = abb^{*}$

- Give examples for words in L

{ab abb abbb …..}

RE-3

- Example

$\sum = \{a, b, c\}$

- Formally describe all words that start with an a followed by any number of b's and then end with c.

$L = a\, b^{*} c$

- Give examples for words in L

{ac abc abbc abbbc …..}

RE-4

- Example

∑ ={a, b}

- Formally describe the language that contains nothing and contains words where any <u>a</u> must be followed by one or more <u>b</u>'s

L = b*(abb*)* OR (b+ab)*

- Give examples for words in L

{Λ ab abb abababb ….. b bbb…}

RE-5

- Example

$\sum = \{a, b\}$

- Formally describe all words where a's if any come before b's if any.

$L = a^* b^*$

- Give examples for words in L

$\{\Lambda\ a\ b\ aa\ ab\ bb\ aaa\ abb\ abbb\ \ bbb\ldots..\}$

NOTE: $a^* b^* \neq (ab)^*$

because first language does not contain abab but second language has.

Once single b is detected then no a's can be added

RE-6

- Example

$\sum = \{a\}$

- Formally describe all words where count of a is odd.

$L = a(aa)^* \text{ OR } (aa)^* a$

- Give examples for words in L

{a aaa aaaaa …..}

RE-7.1

- Example

$\sum$ ={a, b, c}

- Formally describe all words where single a or c comes in the start then odd number of b's.

$L = (a+c)b(bb)^*$

- Give examples for words in L

{ab cb abbb cbbb …..}

RE-7.2

- Example

$\Sigma = \{a, b, c\}$

- Formally describe all words where single a or c comes in the start then odd number of b's in case of a and zero or even number of b's in case of c.

$L = ab(bb)^* + c(bb)^*$

- Give examples for words in L

{ab c abbb cbb abbbbb …..}