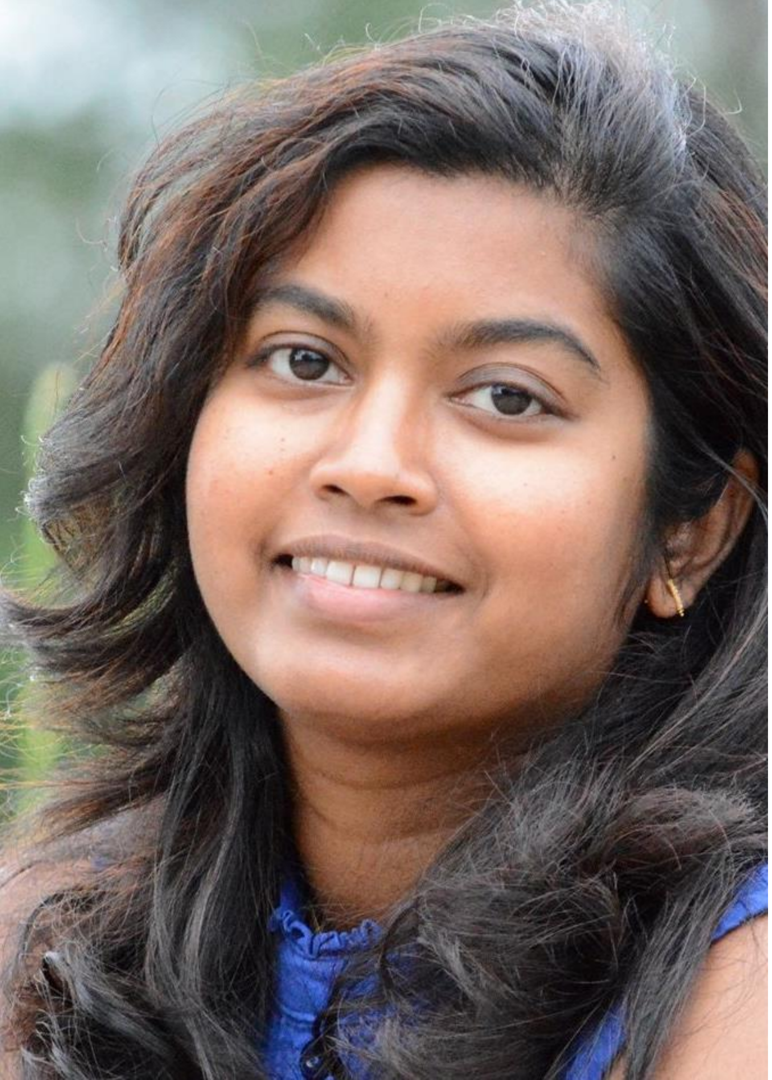
SCS 2112 : Automata Theory

Dinuni Fernando, PhD



Dr. Dinuni Fernando

- Holy Family Convent – Colombo 4
- Graduate from UCSC 2014 – CS major, 4 year degree, 1st class honors
- PhD – SUNY Binghamton, NY, USA 2019
- Joined UCSC 2019 as a Senior Lecturer
- Research Interests
  - Virtualization – Cloud computing
  - Blockchain and security
  - Software-defined networking

Class Logistics

- Slides can be found in LMS
- Recommended reading
    - Fundamentals of the Theory of Computation, Principles and Practice, Raymond Greenlaw , H James Hoover, Morgan Kaufmann, 2013
    - An Introduction to Formal Languages and Automata, Peter Linz, Narosha Publishing House.
    - Introduction to Automata Theory, Languages and Computation, John E. Hopcroft, Rajeev Motwani, Jeffrey D Ullman, Pearson Education
- Evaluation Criteria : 60% paper, 40% Assignments / Quizes/ Tutes

Automata Theory

- Study of abstract computing devices or machines.
- To model the hardware of a computer, notion of an automaton is introduced. An automation is a construct that possesses all the indispensable features of a digital computer.
    - eg: Accepts input, produces output, may have temporary storage and can make decisions in transforming the input into the output.

Mathematical Preliminaries and Notation

1. Sets : collection of objects
   a. May contains different types of objects
   b. Order of objects are not important

      $X \in S$: X is an element of set S
      $X \notin S$: X is not in S

   Eg: set of integers $S = \{0,1,2\}$

   Eg: $T = \{i : i > 0, i \text{ is even}\}$: T is the set of all i, such that i is greater than zero, and i is even

Mathematical Preliminaries and Notation

2. Sequence : collection of object in a particular order
    1. May contain different types of objects
    2. Order is crucial
3. Tuples : A sequence of finite number of objects (finite ordered list)

    eg: k tuple : sequence of k elements

UCSC

Mathematical Preliminaries and Notation

4. Pair : 2 tuple
5. A x B : cartesian product of A and B

$A \times B = \{ (a,b) | a \in A \text{ and } b \in B\}$

Eg: Let $A = \{1,2\}$, $B = \{x,y\}$

$A \times B = \{ [1,x], [1,y], [2,x], [2,y]\}$

SET Operations

- Union (∪)

$S_1 \cup S_2 = \{x: x \in S_1 \text{ or } x \in S_2 \}$

- Intersection (∩)

$S_1 \cap S_2 = \{x: x \in S_1 \text{ and } x \in S_2 \}$

- Difference (-)

$S_1 - S_2 = \{x: x \in S_1 \text{ and } x \notin S_2 \}$

- Complementation (S bar / $\bar{S}$)

$\bar{S} = \{x: x \in U \text{ and } x \notin S \}$, U is the universal set that all possible elements belong.

Sets (cont'd)

- Empty set / null set ϕ – set with no elements.
- Subset - A set S_1 is said to be a subset of S if every element of S_1 is also an element of S.

  $$S_1 \subseteq S$$

- Proper subset : if $S_1 \subset S$, but S contains an element not in S_1 , we say that S_1 is a proper subset of S.
- Disjoint set : if S_1 and S_2 have no common element, that is, $S_1 \cap S_2 = \emptyset$

UCSC

Sets (cont'd)

- **Finite** set : set with finite number of elements. Else **infinite.**
- The size of a finite set is the number of elements in the set. Denoted by |S|.
- Powerset : the set of all the subsets of a set S. Denoted by 2^S.
- Eg: S is the set {a,b,c}, then its powerset is

$$2^S = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$$

Here |S| = 3 and $| 2^S |$ = 8. This is an instance of a general result; if S is finite, then $| 2^S | = 2^{|S|}$

UCSC

Functions

Function is a rule that assigns to elements of one set a unique element of another set. If *f* denotes a function, then first set is called the *domain* of *f*, and the second set is its *range*.

$f : S_1 \rightarrow S_2$

$f : D \rightarrow R$

- Domain of f is D
- Range of f is R

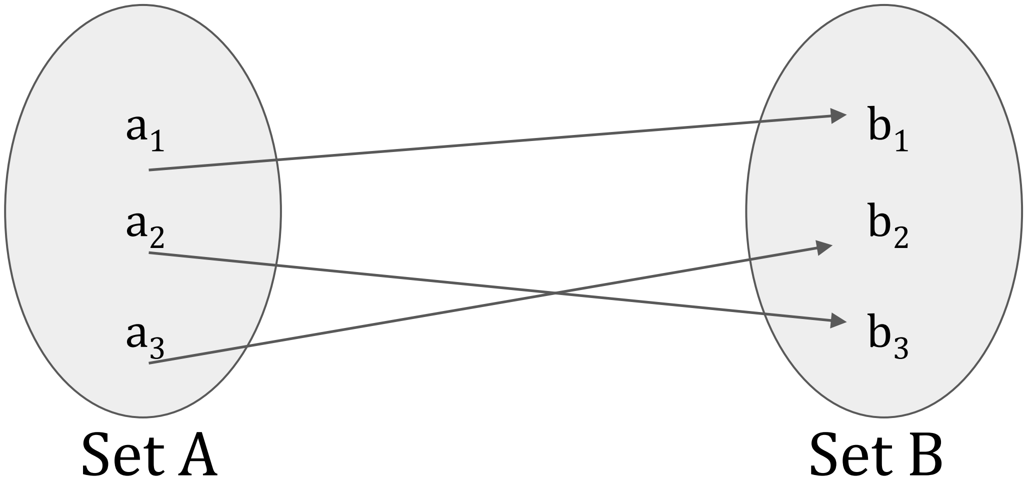
Types of functions

- Onto : A function that uses all values of the range
- Into : A function that does not use all values of the range
- 1-1

Types of functions

1. One-one Function or Injective Function

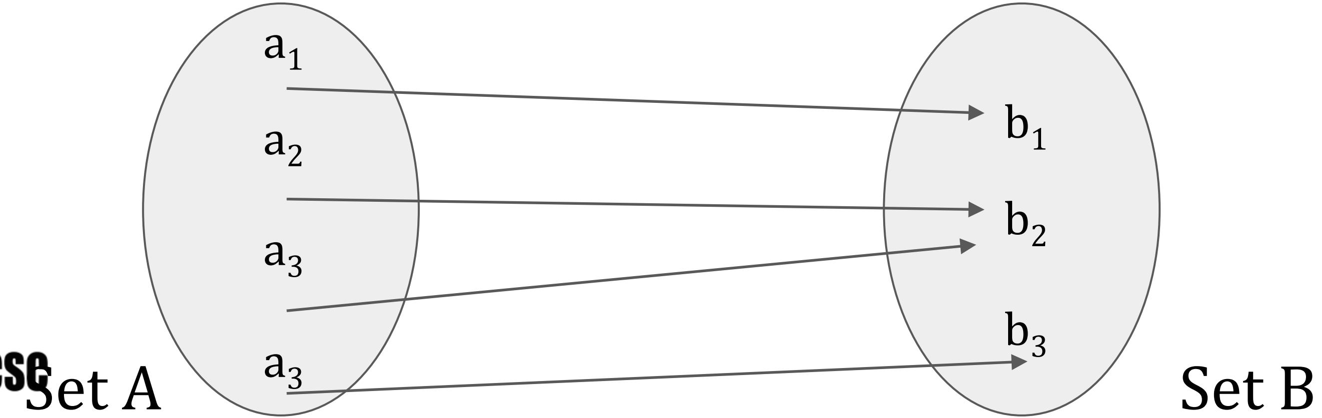
If each elements of set A is connected with different elements of set B.



Types of functions

2. Onto function or Surjective function

Function f from set A to set B is onto function if for every element of B there is at least one or more than one element matching with A.

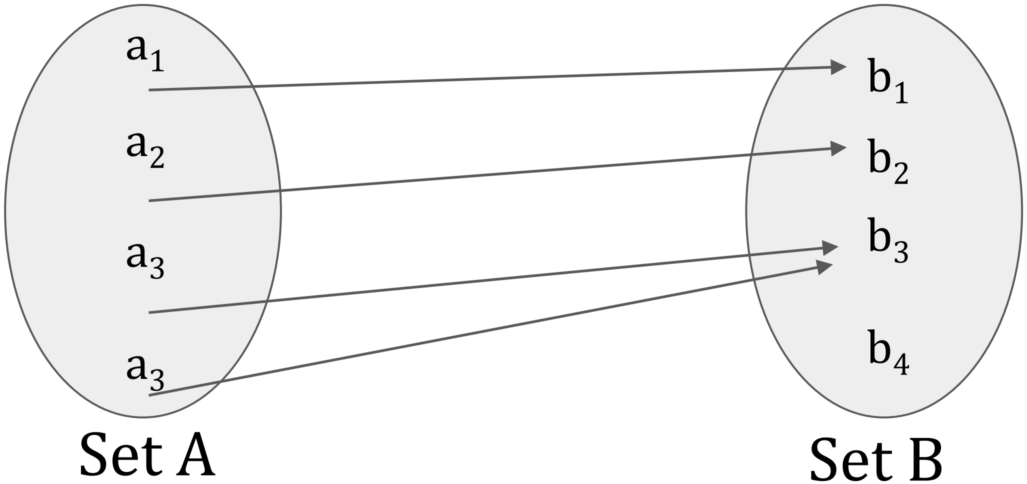


UCSC

Types of functions

3. Into Function

   Function f from set A to set B is Into function if at least set B has a element which is not connected with any of the element of set A.



Languages

- Natural Languages
  - Difficult to define
  - Dictionary Definition : System suitable for expressing ideas, facts or concepts and rules for their manipulation.
- Formal Languages
  - Defined precisely so that mathematical analysis is possible.
- Two basic problems in programming language design are
  - How to define a programming language precisely ?
  - How to use such definitions to write an efficient and reliable translation programs ?
- Theory of formal languages are extensively used in the
  - Definition of programming languages.
  - Construction of interpreters and compilers.

UCSC

Languages : Alphabet / String

- Alphabet : Finite non-empty $\sum$ set of symbols
- Strings : finite sequences of symbols from the alphabet

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

Strings = ab, aba, abba,....

Two strings are considered the same if all their letters are the same and in the same order.

UCSC

String operations

- Concatenation

  $w = a_1a_2a_3\ldots,a_n$; $v = b_1b_2b_3\ldots,b_m$; $wv = a_1a_2a_3\ldots,a_nb_1b_2b_3\ldots,b_m$
- Reverse of a string

  $w = a_1a_2a_3\ldots,a_n \Rightarrow w^R = a_na_{n-1}a_{n-2}\ldots,a_1$
- Length of a string |w| : $L^1 = L$
- Sub-string : Any string of consecutive characters in some sitting w.
- w='ab' , possible substrings == {ε, a,b,ab}
- Empty string denoted by λ : $L^0 = \{ \lambda, \varepsilon \}$

String operations

- Prefix / suffix
  - w = 'ab', prefix p(w) = {λ ,a,ab} , suffix s(w) = {λ,b,ab}
  - ∑* : Kleene closure : strings obtained by concatenation zero or more symbols from ∑. Always contain λ (ε)
  - ∑+ = ∑* - {λ}
  - ∑* , ∑+ are infinite
- Informally a language L over an alphabet is a subset of ∑*.
  - ∑* , ∑+ are infinite

Since a language is a set, all set operations can be applied on languages.

String operations

- Since languages are sets, the union, intersection, and difference of two languages are automatically defined.
- The complement of a language is defined with respect to Σ^*.

$$L' = \Sigma^* - L$$

- Concatenation of two languages L1 and L2 contains every string in L1 concatenated with every string in L2.

$$L1L2 = \{xy \mid x \in L1, y \in L2\}$$

UCSC

String operations

L^n is defined as the concatenation of L with itself n times

$L^0 = \{\lambda\}$

$L^1 = L$

The star-closure of a language is defined as

$$L^* = L^0 \cup L^1 \cup L^2 \ldots\ldots$$

The positive closure of a language is defined as

$$L^+ = L^1 \cup L^2 \ldots\ldots$$

A string in a language L is called a **sentence** of L.

UCSC

How specific languages can be defined?

- Listing out all possible words in the language, if the language is finite.
  - Eg: a Dictionary
- Giving a set of rules, which defines all the acceptable words of the language.
- A language L over an alphabet is a subset of $\sum^*$. Thus set notations can be used to define languages. However set notation is inadequate to define complex languages.
- Grammars : Powerful mechanism for defining formal languages.

Formal Languages

- Alphabet (∑) : A finite nonempty set of symbols.
- Syntax : linguistic form of sentences in the language

  Only concerned with the form rather than meaning

- Semantics : Linguistic meaning of syntactically correct sentences.

  A syntactically correct program need not make any sense semantically.

Formal definition of grammar

Grammar G is defined as a quadruple : G = (V,T,S,P)

V/N : finite set of objects called variables (non-terminals, denoted by capital letters)

T : finite set of objects called terminal symbols

S : Initial non-terminal deriving symbol/ start symbol (S $\in$ V, N and T are non-empty and disjoint)

P : finite set of production

Formal definition of grammar

All production rules are of the form of $x \to y$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$.

Given a string w : *w= uxv*

production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string z

$$z = uyv$$

$w \Rightarrow z$; *w derives z / z is derived from w*

UCSC

Formal definition of grammar

We may derive new string from a given string by applying productions successively in arbitrary order.

S ⇒w1 ⇒ w2 ⇒ …….. ⇒ wn ⇒ *w : derivation of sentence w*

This can be given as w1 ⇒ * wn

this means w1 **derives** wn

w1, w2,……. wn are called **sentential forms** of the derivation.

UCSC

Sentential Form

A sentential form is the start symbol S of a grammar or any string in $(V \cup T)^*$ that can be derived from S.

Consider the following linear grammar

$G = (\{S,B\},\{a,b\},S,\{S \longrightarrow aB,\ S \longrightarrow B,\ B \longrightarrow bB,\ B \longrightarrow \lambda\})$

Derivation of the above grammar :

$S \longrightarrow aB \longrightarrow abB \longrightarrow ab\ /\ abbB \longrightarrow abb$

UCSC

Example 1A

Consider the following linear grammar

G = ({S,B},{a,b},S,{S⟶aS, S⟶B, B⟶bB, B⟶λ})

Derivation of the above grammar :

$S \Rightarrow aS \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abb$

Each of {S, aS, aB, abB, abbB, abb} is a sentential form.
Because this grammar is linear, each sentential form has at most one variable. Hence there is never any choice about which variable to expand next.

Grammars

Let G be a grammar. Then the language generated by G is denoted by L(G).

Two grammars are said to be equivalent if they generate the same language.

- Important in the development of parsers.
- It is hard/impossible to develop parsers for some grammars.
- They may be transformed into equivalent grammars that can be parsed.

Example 1B

- The set of all legal identifiers in Pascal is a language.
- Informal Definition : Set of strings with a letter followed by an arbitrary number of letters or digits.

Formal Definition : (Grammar)

<id> ⟶ <letter><rest>

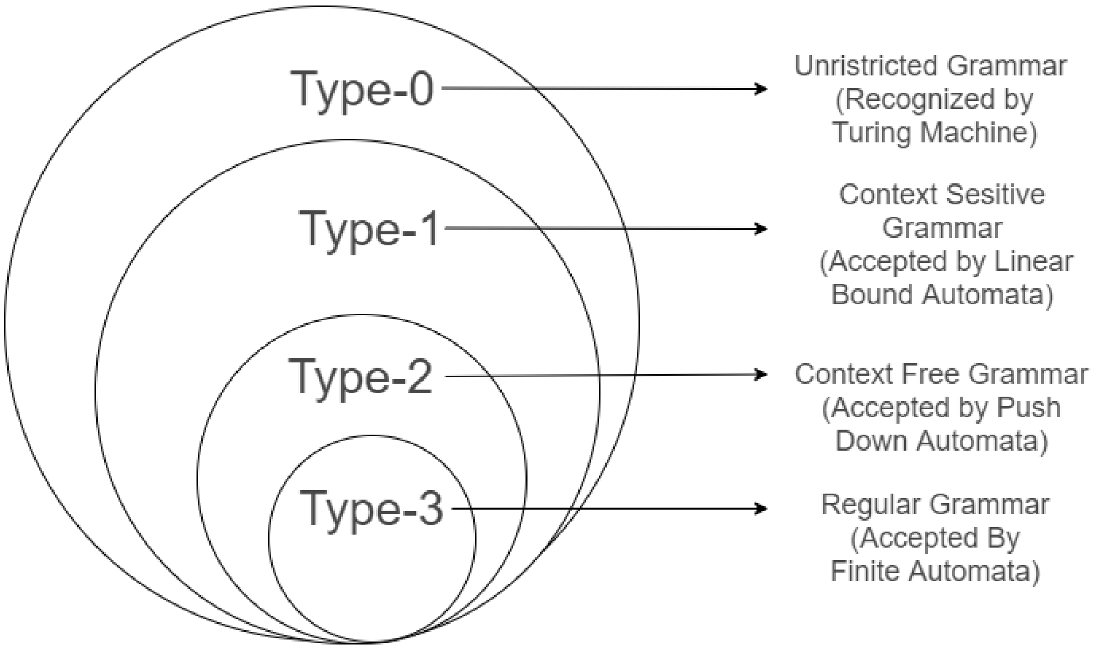
<rest> ⟶ <letter><rest> | <digit><rest> | ∈

<letter> ⟶ a | b| c|.....|z

<digit> ⟶ 0|1|.....|9

UCSC

Chomsky scheme of grammar classification



A language L(G) is said to be of type k if it can be generated by type k grammar.

Chomsky scheme of grammar classification

- Type 0 : Unrestricted grammars
  - Includes all formal grammars.
  - Also known as Recursive enumerable languages.

Production in the form of $\alpha \longrightarrow \beta$

$\alpha = (V + T)^* V (V + T)^*$, where V are non-terminals and T are terminals

$\beta (V + T)^*$

In type 0, there must be at least one variable on left side of production.

Eg: $A \longrightarrow S$

$Sab \longrightarrow ba$, Here A, S are non-terminals(variables), a,b are terminals.

Chomsky scheme of grammar classification

- Type 1 : Context sensitive grammars

  Production in the form of $\alpha \longrightarrow \beta$

  $|\alpha_i| <= |\beta_i|$ for all i , where | | denotes the length

  Note : null string would not be allowed as a right hand side of any production.

  All languages can be detected by linear push down automata.

$$S \longrightarrow AB$$
$$AB \longrightarrow abc$$
$$B \longrightarrow b$$

UCSC

Chomsky scheme of grammar classification

- Type 2 : Context free grammars (BNF Grammars)

  $A \longrightarrow y$; A is a non-terminal , y is a string with terminals and non-terminals .

  Languages can be exactly recognized by non-deterministic pushdown automata

  Context free grammar is a common notation for specifying the syntax of programming languages.

  eg : In C if-else statement

  stmt $\longrightarrow$ if (expr) stmt else stmt

$S \longrightarrow AB$

$A \longrightarrow a$

$B \longrightarrow b$

Chomsky scheme of grammar classification

- Type 3 : regular grammar
  - All production of the form $A \longrightarrow xB$ or $A \longrightarrow x$ where A and B are non-terminals and x is in $\sum^*$ - right linear grammar.
  - All production of the form $A \longrightarrow Bx$ or $A \longrightarrow x$ where A and B are non-terminals and x is in $\sum^*$ - left linear grammar.
  - Can be recognized by finite automata .
  - Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

Chomsky scheme of grammar classification

- Type 3 : regular grammar shoud be either left or right regular grammar

$N \longrightarrow NT$
$N \longrightarrow T$
N – non terminls / T – terminals

Left regular grammar

$N \longrightarrow TN$
$N \longrightarrow T$
N – non terminls / T – terminals

Right regular grammar

Exercises

1. G=({S,A},{a,b},S,P) , write sentences generated by the following grammar

   a) $S \longrightarrow Ab$

      $A \longrightarrow aAb$

      $A \longrightarrow \lambda$

UCSC

Exercises

1. G=({S,A},{a,b},S,P) , write sentences generated by the following grammar

   b) $S \longrightarrow SS$

   $S \longrightarrow \lambda$

   $S \longrightarrow aSb$

   $S \longrightarrow bSa$

Exercises

2. $G=(\{S,A\},\{a,b\},S,P)$, write sentences generated by the following grammar and provide the language.

a) $S \longrightarrow aAb \mid \lambda$

$A \longrightarrow aAb \mid \lambda$

b) $S \longrightarrow aA$

$A \longrightarrow bS$

$S \longrightarrow \lambda$

UCSC