## Computer Theory

Background

This topic deals with things you can compute mechanically, how fast and how much space it takes to do so.

People like Aslonzo Church, Stephen Kleene, Emi l Post, Andrei Andreevich Markov, John von Neumann, and Alan Turing independently came up with building blocks for mathematical algorithms.

This topic deals with machines that use rules to accept or reject input.

**FSM**: Finite State Machine

Simplest computation model

Limited memory (cannot store or count strings)

**CFL:** Context Free Language

Higher level computation model

**Turing machine:**

High level computation model

**Undecidable:**

Problems that cannot be solved mechanically

Undecidable

Turing Machine

CFL

FSM

**Lesson 0**

Languages

Finite State machine/ Finite Automata (Prerequisites) **S-A-S-L**

**Symbol**

Example a,b,c,0,1,2,3

**Alphabet**: Collection of symbols

Example

**String/Word**: These are any nonempty strings of alphabet characters

Example

In , etc are words.

**Language**: Set of strings

Example

Set of all strings of length 2

Example

Set of all strings that begin with a

infinite language

Powers of Σ

Example: considering the alphabet

Set of all strings of length 0

or empty set

Set of all strings of length 1

Set of all strings of length 2

Set of all strings of length n

Infinite set

Cardinality: Number of elements in a set

1 element

2 elements

4 elements

based on 2 elements in alphabet

(Sigma star/Kleene Closure/Kleene Star)

set of all possible strings of all lengths over

infinite set

(infinitely many words, each of finite length.)

**Lesson 1**

Recursive Definitions

This is a method to define a language

Recursive Enumerable (RE) Languages

Can be accepted/recognized by Turing Machine (Turing Recognizable language)

It will enter the final state for the strings of language.

It may/may not enter rejecting state for strings not in the language.

Recursive Language (REC)

Can be decided by Turing Machine (Turing Decidable Language)

It will enter the final state for the strings of language.

It will not enter rejecting state for strings not in the language.

RE

REC

Why do we need to define sets using a recursive definition?

Because sometimes the number of elements in a set are infinite, and a recursive definition gives us a finite definition of **infinite sets**

**How to write a recursive definition**

|  |  |
| --- | --- |
| Recursive Definition | Clause |
| Specify some of the basic elements in the set. | Basis Clause |
| Give some rules for how to construct more elements in the set from the elements that we know are already there. | Inductive Clause |
| Say that there are no other elements in the set except those constructed using steps 1 and 2. | Extremal clause |

**Example: EVEN (normal recursive definition)**

1. 2 is in EVEN
2. If x is in EVEN, and y is in EVEN, x+y is in EVEN.
3. The only elements in the set EVEN are those that can be produced from RULEs 1 and 2 above

**Example: POLYNOMIAL (normal recursive definition)**

1. Any number is in POLYNOMIAL.
2. The variable x is in POLYNOMIAL.
3. If p and q are in POLYNOMIAL, then so are p + q, p - q, (p), and pq.

**Example: ODDnotAB (normal recursive definition)**

1. ODDnotAB is the smallest subset of {a b}\* such that a, b ∈ ODDnotAB
2. if w ∈ ODDnotAB, then also CONCAT( bb, w ) ∈ ODDNOTAB

and CONCAT(w,aa) ∈ ODDNOTAB

…TODO

**Example: ODDnotAB (Cohen’s recursive definition)**

Rule 1: *w* ∈ AB.

…TODO

**Lesson 2**

Inductive Proofs

This is a method to define a language

<https://www.mathsisfun.com/algebra/mathematical-induction.html>

This essentially is the same as using a recursive definition to define a language. It uses a trick which allows you to prove a statement about an arbitrary number n by first proving it is true when n is 1 and then assuming it is true for n=k and showing it is true for n=k+1.

**How to write a proof by induction:**

|  |  |
| --- | --- |
| Recursive Definition | Clause |
| Show n=1 is true (LHS = RHS) | Basis Clause |
| substitute k into formula (LHS) | Inductive Clause |
| Substitute k+1 into formula (RHS) | Extremal clause |

**Example:** Prove 1+2+...+n=n(n+1)/2

**Lesson 3**

Regular languages

A language is said to be regular if and only if some finite state machine recognizes it.

A language is not regular if it not recognized by a finite state machine or require memory.

Example

The language has a repeating pattern of ababb.

We cannot store this pattern.

This language is not regular.

Example

The language has the same number of a’s as it does b’s.

We cannot store the number of a’s.

This language is not regular.

Operations on regular languages

Example

**Union**

**Concatenation**

**Star**

infinite set

Theorems

The class of Regular languages is closed under UNION.

i.e. is also a regular language

The class of Regular languages is closed under CONCATENATION.

i.e. is also a regular language

**Lesson 4**

Regular Expressions

This is a method to define a language

|  |  |
| --- | --- |
| **Character** | **Meaning** |
| **\*** | Kleene Star. Unknown or undetermined power. Can be empty |
| **+** | Or. |
|  | Empty set. |
| **()** | Group. Can be empty |
| **[]** | Range. Cannot be empty |

Notes:  
 can always be

can always be

**Lesson 5**

Kleene’s Theorem

Any language that can be defined by

regular expression, or

finite automaton, or

transition graph

can be defined by all three methods.

**Lesson 6**

Finite Automata

This is a method to define a language

FA without output

FA without output

Moore

Machine

Mealy

Machine

NFA

NFA

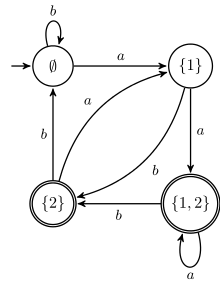
DFA

**FSM**: Finite State Machine

Simplest computation model

Limited memory

How to draw Finite Automata



Circles: States

Double-Circles: Terminating State

Edges: Transitions

Edge-labels: Inputs

How to represent Finite Automata

Q Finite Set of all states

Σ Inputs: An alphabet

Start state/initial state

Final state

Transition function, maps

Example

**A close up of a blackboard

Description automatically generated**

Q

Σ

Transition function, maps

(i.e. shows product of states with all inputs)

Transition function

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | C | B |
| B | D | A |
| C | A | D |
| D | B | C |

**Lesson 7**

Deterministic Finite Automata

**Lesson 8**

Non-Deterministic Finite Automata

Using Kleene’s theorem, lets convert an FA to a DFA:

Example

**FA**

Set of all strings that start with ‘0’

**DFA**

**A picture containing object, clock

Description automatically generated**

digraph finite\_state\_machine {

rankdir=LR;

size="8,5"

node [shape = circle]; A;

node [shape = doublecircle] B;

node [shape = circle]; C;

node [shape = point ]; Ai;

Ai -> A;

A -> B [ label = "0" ];

A -> C [ label = "1" ];

C -> C [label = "0,1"];

B -> B [label = "0,1"];

}

(Where C is a dead state)

Example: Construct a DFA that accepts all strings over {0,1} of length 2

**FA**

Set of all strings that start with ‘0’

**A close up of a clock

Description automatically generatedDFA**

digraph finite\_state\_machine {

rankdir=LR;

size="8,5"

node [shape = circle]; A;

node [shape = circle] B;

node [shape = doublecircle]; C;

node [shape = circle] D;

node [shape = point ]; Ai;

Ai -> A;

A -> B [ label = "0,1" ];

#C -> C [label = "0,1"];

B -> C [label = "0,1"];

C -> D [label = "0,1"];

D -> D [label = "0,1"];

}

(Where D is a dead state)

Example: Construct a DFA that accepts any strings over {a,b} that does not contain the string aabb in it

**Simplify the problem:**

Construct a DFA that accepts any strings over {a,b} that contains the string aabb in it

Minimization of DFA

A close up of a watch

Description automatically generatedExample: Transition table:

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
| A | B | C |
| B | B | D |
| C | B | C |
| D | B | E |
| E | B | C |

Write down state(s) together as a set and final state(s) as a set

**0 Equivalence:** {A,B,C,D} {E}

Check the transition table, which states are similar. Remove any that point to the final state

**1 Equivalence:** {A,B,C} {D} {E}

Check the transition table, which states are similar. Remove any that are not similar

**2 Equivalence:** {A,C} {B} {D} {E}

**A picture containing object, clock, drawing

Description automatically generated**