## Computer Theory

Background

This topic deals with things you can compute mechanically, how fast, and how much space it takes to do so. The aim is to construct regular language acceptor machines (finite automata, Moore and Mealy machines, transition

graphs and nondeterministic finite automata), which are simple abstract models of computers.

People like Alonzo 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 (R-C)

This is a method to define a language

A description of a set is a recursive definition if it characterises that set as the subset of some universal set that is the smallest of all those subsets which contain a certain starting point and are closed

under some given function.

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**

**1 - Normal Recursive Definition (Cohen’s Style)**

|  |  |
| --- | --- |
| 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/Cohen’s style)

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/Cohen’s style)

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/Cohen’s style)

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

Example: ASS2 Q1

ODDnotAB, over the alphabet

**The equipment (UGF)**

*Define the basic equipment to enable us to form a recursive definition*

|  |  |  |
| --- | --- | --- |
| Requirement | Explanation |  |
| Universal set | the set containing all objects or elements and of which all other sets are subsets | The set {a b}\* will be suitable because it contains, along with other words, all the words in the language ODDnotAB |
| Generator | the shortest words in the set | The generators are a and b |
| Function | a function defined on the universal set. | The function CONCAT as defined in learning unit 3 will be suitable (This is the function we will use to add letters to the generators above) |

**How to use the equipment**

*Use the function to form new words, while avoiding constraints*

*(does not start with a or end in b)*

**Part 1: Recursive definition** (using equipment)(**U-F-G**)

ODDnotAB is the smallest subset of {a b}\*, *Universal Set*

such that such that a, b ∈ ODDnotAB *Generators*

If Q ∈ ODDnotABthen

CONCAT(bb, Q) ∈ AB, *Function*

CONCAT(Q, aa) ∈ AB

If Q ∈ ODDnotAB andQ **does not end on an a then**

CONCAT(Q, ba) ∈ AB,

CONCAT(Q, bb) ∈ AB

If Q ∈ ODDnotAB and Q **does not begin with b then**

CONCAT(aa, Q) ∈ AB,

CONCAT(ba, Q) ∈ AB

**Part 2: Cohen’s recursive definition**(**F-G**)

Rule 1: a, b ∈ ODDnotAB.

*(add two letters at a time using CONCAT. Remember to write the constraints)*

Rule 2: If Q ∈ ODDnotABthen

CONCAT(bb, Q) ∈ AB,

CONCAT(Q, aa) ∈ AB

If Q ∈ ODDnotAB andQ **does not end on an a then**

CONCAT(Q, ba) ∈ AB,

CONCAT(Q, bb) ∈ AB

If Q ∈ ODDnotAB and Q **does not begin with b then**

CONCAT(aa, Q) ∈ AB,

CONCAT(ba, Q) ∈ AB

Rule 3: Only words generated by rules 1 and 2 are in ODDnotAB.

**Lesson 2**

Inductive Proofs (C-I-P)

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.

**1 - 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:** Example exam solution

**Q1: Cohen’s recursive definition**

Give a recursive definition of the set P of all integers greater than or equal to 5

**Q2: Induction Principle**

Formulate an appropriate induction principle

**Q3: Proof by Induction**

Apply the induction principle to prove that for all integers n ≥ 5.

*define A* **⊆** *Q. A = LHS = RHS*

*Goal1: prove that A* **⊆** *Q i.e. sub 1, prove LHS = RHS*

*Goal2: assume that k* **⊆** *A. i.e. sub k*

*Goal3: prove that* k + 1 ∈ A **⊆** *Q i.e. sub k+1 into LHS, prove it’s equal to sub k+1 into RHS*

**Q1: Cohen’s recursive definition**

P is the smallest subset of Z(the set of integers) such that

5 P, and if

n P, then also

n + 1 P

**Q2: Induction Principle**

If a subset A of P, is such that

5 A, and if

n A, then also

n + 1 A, then

A = P

**Q3: Proof by Induction**

* Show for n = 5 n = 1
* Assume k A k, LHS
* Prove k + 1 A k + 1, RHS

Thusk + 1 A Q1

Hence, A = P. Q2

We can conclude that for all integers n ≥ 5

Example: ASS2 Q2

Q1 Give a recursive definition of the set P of all positive integers greater than 0,

Q2 formulate the appropriate induction principle, and then

Q3 use mathematical induction to prove that

11 + 15 + 19 + … + (4n + 7) = + 9n

for all positive integers n > 0.

**Q1: Cohen’s recursive definition**

Rule 1: 1 ∈ Q.

Rule 2: If x ∈ Q then x+1 ∈ Q.

Rule 3: Q is the smallest set satisfying R1 and R2.

**Q2: Induction Principle**

*(same as above, to create the subset A)*

If a subset A of Q such that 1 ∈ A then also

k + 1 ∈ A, then A = Q

**Q3: Proof by Induction**

*(define A* **⊆** *Q. A = LHS = RHS)*

A = n| n ∈ Q and = + 9n

*(Goal1: prove that A* **⊆** *Q) (1 ∈ A)*

LHS = (4n + 7) RHS = + 9n

= 4(1) + 7 = + 9(1)

= 11 = 11

LHS = 11 = RHS, thus 1 ∈ A

*(Goal2: assume that k* **⊆** *A) (k ∈ A)*

A = n| n ∈ Q and = + 9k

*(Goal3: prove that* k + 1 ∈ A **⊆** *Q)*

*(sub k + 1 into LHS)*

Assume k + 1

= ( + 9k) +

= + 9k + 4k + 4 + 7

= + 4k + 2 + 9k + 9 group differently

= + 2k + 1) + 9(k + 1)

= 2 + 9(k + 1)

Thus k + 1 ∈ A Q1

Hence, A = P. Q2

We can conclude that = + 9n for all integers n > 0

**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

This is describing a language by giving a very simple kind of rule that specifies how the strings belonging to the language can be built up

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

Notes:  
 can always be

can always be

What is the difference between Λ and ?

Λ - **ϵ is a word**. represents the set that contains only the empty string {ε}

|{ϵ}|=1

– **is a language.** represents the empty set of strings ∅={}.

|∅|=0

**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.

This can be further expanded using the basic definition of Regular Expression, where:

Say, and be two regular expressions. Then,

1. is a regular expression too, whose corresponding language is (UNION)
2. is a regular expression too, whose corresponding language is

(PRODUCT language)

1. is a regular expression too, whose corresponding language is

A picture containing clock

Description automatically generated

Intersection of Regular languages:

1. is a regular expression too, whose corresponding language is (INTERSECT). The set of regular languages is closed under intersection.

**Lesson 6**

FA: With Output

**FSM**:

Simplest computation model

Limited memory

FA without output

DFA

NFA

NFA

FA without output

Moore

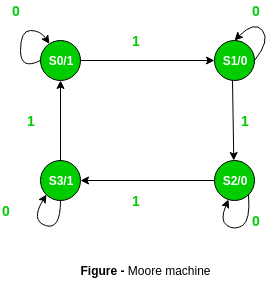
Machine

Mealy

Machine

|  |  |
| --- | --- |
| **Mealy Machine** | **Moore Machine** |
| Output depends both upon the present state and the present input | Output depends only upon the present state. |
| Generally, it has fewer states than Moore Machine. | Generally, it has more states than Mealy Machine. |
| The value of the output function is a function of the transitions and the changes, when the input logic on the present state is done. | The value of the output function is a function of the current state and the changes at the clock edges, whenever state changes occur. |
| Mealy machines react faster to inputs. They generally react in the same clock cycle. | In Moore machines, more logic is required to decode the outputs resulting in more circuit delays. They generally react one clock cycle later. |
| Difficult to design | Easy to design |

**Moore Machine** – A Moore machine is defined as a machine in theory of computation whose output values are determined **only by its current state**.

Outcome is attached directly to the state

It has 6 tuples: (Q, q0, ∑, O, δ, λ)

Q is finite set of states

q0 is the initial state

∑ is the input alphabet

O is the output alphabet

δ is input transition function which maps Q×∑ → Q

λ is the output function which maps Q → O

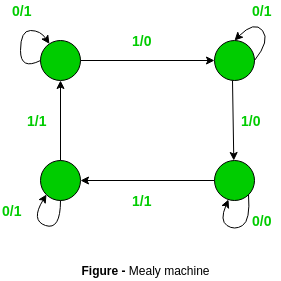
|  |  |  |  |
| --- | --- | --- | --- |
| Present State | Next State | | output |
| Input = 0 | Input = 1 |
| → a | b | c | x2 |
| b | b | d | x1 |
| c | c | d | x2 |
| d | d | d | x3 |

A picture containing object, photo, clock, looking

Description automatically generated

**Mealy Machine** – A Mealy machine is defined as a machine in theory of computation whose output values are determined by both its current state and current inputs. In this machine at most one transition is possible.

Outcome/output depends on where you have come from

It also has 6 tuples: (Q, q0, ∑, O, δ, λ’)

Q is finite set of states

q0 is the initial state

∑ is the input alphabet

O is the output alphabet

δ is transition function which maps Q×∑ → Q

‘λ’ is the output function which maps Q×∑→ O

**Lesson 7**

Transition Graphs

A picture containing clock

Description automatically generatedExample: ASS3 Q2

- indicates a start state

+ indicates an end state

Remember to separate the above from actual states/circles

|  |  |  |
| --- | --- | --- |
| **New State** | **Read an a** | **Read a b** |
|  |  |  |
|  |  |  |
|  |  |  |
| or |  |  |
| or or |  |  |

Example: ASS3 Q3

Moore Machines: What is the output if the string is abbbaabab?

**A picture containing photo, snow, table, hanging

Description automatically generated**

**Input Output**

start q0 -> 0

a q1 -> 1

b q3 -> 1

b q3 -> 1

b q3 -> 1

a q1 -> 1

a q0 -> 0

b q2 -> 1

a q1 -> 1

b q3 -> 1

Example: Convert Moore to Mealy

**State Output**

-> A b

B b

C a

**A screen shot of a computer

Description automatically generated**

**Lesson 9**

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**

**Exam Curriculum:**

Languages: L0 S-A-S-L

Regular Expressions: L4

TODO:

Provide a regular expression for a required language

Analyse a regular expression to determine the language generated.

**Math tools:**

~~Recursive Definitions: L1~~ R-C

~~Mathematical Induction: L2~~  C-I-P

**Machines:**

FAs (Finite Automata),

TGs (Transition Graphs),

NFAs (Non-deterministic Finite Automata),

Language Acceptors (Mealy and Moore machines)

TODO:

How to draw each machine

Determine language being accepted by machine

Determine output of Mealy Machine

Determine output of Moore Machine

Convert Mealy to Moore or vice versa

Draw Mealy to Moore given the output of a machine

Kleene’s Theorem: TG -> RE, RE -> FA

Pumping Lemma with Length

Decidability