## 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**

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 2**

Recursive Definitions: Decidability

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.

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**

**Recursive Enumerable (RE) Languages:** Turing Recognizable language

A language ‘L’ is said to be recursive if there exists a Turing machine:

* which will accept all the strings in ‘L’
* reject the strings not in ‘L’
* It may/may not enter rejecting state for strings not in the language.

*it may or may not halt every time and give an answer (accepted/rejected) for very string input.*

**Recursive Language (REC):** Turing Decidable language

A language ‘L’ is said to be recursive if there exists a Turing machine:

* which will accept all the strings in ‘L’
* reject the strings not in ‘L’
* It will not enter rejecting state for strings not in the language.

*it will halt every time and give an answer (accepted/rejected) for very string input.*

**A picture containing game, sport

Description automatically generated**

RE

REC

**Decidable Language**

A language ‘L’ is decidable if it is a recursive language

All decidable languages are recursive languages and vice-versa

*The Turing machine will always halt*

**A language**

A language ‘L’ is partially decidable if ‘L’ is a RE language

*The Turing machine may/may not halt*

**Undecidable Language**

A language ‘L’ is undecidable if it is not decidable

An undecidable language may sometimes be partially decidable but not decidable

If a language is not even partially decidable, then there exists no Turing machine for that language

**Lesson 3**

Recursive Definitions

**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 4**

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 5**

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 6**

Pumping Lemma with length

This is used to prove that a language is not regular

*It cannot be used to prove that a language is regular*

If A is a Regular language, then A has a pumping length P such that any string S, where S may be divided into 3 parts S = xyz such that the following conditions must be true:

**Pumping conditions**

* for every i 0

*If y is increased any number of times, must still belong to A*

* |y| > 0

*Length of y must be greater than 0*

* |xy| P

*Length of x and y must be less than the pumping length*

To prove that a language is not Regular using pumping lemma, we prove using contradiction

Remember, Proof by contradiction (*reductio ad absurdum)*

Our goal is to prove S

Assume S

Prove a contradiction

**Proof by contradiction**

Assume that A is regular, then:

It must have a pumping length P

All strings longer than P can be pumped |S| P

Find a string S in A such that |S| P

Divide S into xyz

Show that for some i

Then consider all ways that S can be divided into xyz

Prove a contradiction

Show that none of these can satisfy all 3 pumping conditions at the same time

S cannot be pumped == Contradiction

Example: Using Pumping Lemma prove that the language is not regular

*Cannot be regular as we would need to store the number of a’s to print the same number of b’s, i.e. Finite State Machines have limited memory and cannot store the number of a’s*

**Proof by contradiction**

Assume that A is regular, then:

Pumping length = P *For instance, P = 7*

S = *For instance, S = aaaaaaabbbbbbb*

Divide S into xyz

Case 1: aaaaaaabbbbbbb y is in the ‘a’ part

x y z

Case 2: aaaaaaabbbbbbb y is in the ‘b’ part

x y z

Case 3: aaaaaaabbbbbbb y is in the ‘a’ and ‘b’ part

x y z

Show that for some i *For instance,*

Case 1: aa[aaaa][aaaa]abbbbbbb

This string is not in A (11 a’s 7 b’s)

Case 2: aaaaaaabb[bbbb][bbbb]b

This string is not in A (7 a’s 11 b’s)

Case 3: aaaaa[aabb][aabb]bbbbb

This string is not in A (not in the format

Show that none of these can satisfy all 3 pumping conditions at the same time

* for every i 0

*Not satisfied (from above)*

* |y| > 0

*Satisfied*

* |xy| P
* *Not satisfied (from below)*

Case 1: x = 2, y = 4, therefore |xy| P

Case 2: x = 9, y = 4, therefore |xy| P

Case 3: x = 5, y = 8, therefore |xy| P

S cannot be pumped, Hence A is not regular

**Lesson 7**

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

Remember:

A regular expression for a language L must be able to generate all the words in L and no other words.

to determine whether the language L is defined by r, you may need to provide a counterexample

Example: Construct the regular expression over the alphabet

Set of all strings of length 2

*write out all possibilities*

*convert language to full reg ex*

*find common terms*

*find common terms*

Set of all strings of at least length 2

*write out all possibilities*

Example: Ass 2 Q3

Construct a regular expression that defines the language M

A: containing all words with either exactly one single a-substring

B: **OR** all words with only ab-substrings

over the alphabet

*(remember, a regular expression for a language L must be able to generate all the words in L and no other words)*

[X] 0: Empty string

[X] 0: Null set

[L1] 0: Counterexamples of A

[L2] 0: Counterexamples of B

[L3,L4] 1: Find smallest words that belong to M

[L5,L6,L7] 2: Think about what prefixes/suffixes are needed

Counterexamples of A

= aa,bb,aaa,bbb,aabb,aab,baa,bbaa…

Counterexamples of B

= aa,bb,aba,ababa…

Smallest words in M

= a

Smallest words in M

= ab

*Before: Any number of b’s*

*At least two a’s, then a b*

Prefix of A

= (b\* + aaa\*b)\*

*After: Any number of b’s*

*A b, at least two a’s*

Suffix of A

= (b\* + baaa\*)\*

*Only a suffix is needed here*

Suffix of B

= (ab)\*

CONCAT(L5,L3,L6,L4,L7)

= (b\* + aaa\*b)\*a(b\* + baaa\*)\* + ab(ab)\*

Example: Ass 2 Q4

Construct a regular expression that defines the language M

A: containing all the words with either exactly one aba-substring

B: exactly one bab-substring but not both aba- and bab- substrings

[L1] 0: Counterexamples of A

[L2] 0: Counterexamples of B

[L3,L4] 1: Find smallest words that belong to M

~~[L5,L6,L7,L8] 2: Think about what prefixes/suffixes are needed~~

L1 = aa,bb,aaa,bbb,ba,baa…

L2 = aa,bb,aaa,bbb,ba,baa…

L3 = aba

L4 = bab

TODO

**Lesson 8**

Finite Automata: FA

Example: ASS2 Q5

Build an FA that accepts the language P, consisting of all words with no aa-substring.

A picture containing table, man, hanging, standing

Description automatically generated

Example: ASS2 Q6

Build an FA that accepts the language consisting of only those words that do not end in ab.

A picture containing map, table, display

Description automatically generated

**Lesson 9**

Finite Automata: Moore and Mealy Machines

*These are essentially DFA’s*

**Transducers:** produce output based on a given input and/or a state using actions

i.e. used to recognise languages

*have no final states*

FA with output

Moore

Machine

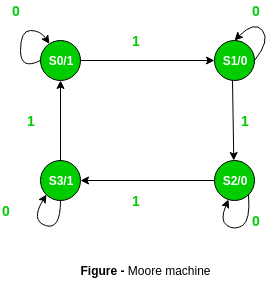
Mealy

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

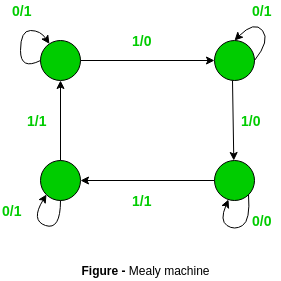
O is the output alphabet

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

λ is the output function which maps Q → O

**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

**Edges are labelled input/output**

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

*Easy: Number of states should be the same*

Construct a Moore Machine that prints ‘a’ whenever the sequence ‘01’ is encountered in any input binary string and then convert it to its equivalent Mealy Machine

*Over the alphabet**. The machine prints ‘b’ in all other instances*

**A picture containing clock

Description automatically generated**

Moore Machine

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **0** | **1** | **Output (associated with current state)** |
| **->**q0 | q1 | q0 | b |
| q1 | q1 | q2 | b |
| q2 | q1 | q0 | a |

**A picture containing map, clock

Description automatically generated**

Mealy Machine: Output is associated to transitions

*Add output of next state to each edge*

*Remove output of attached to each stage*

|  |  |  |
| --- | --- | --- |
| **State** | **0** | **1** |
| **->**q0 | q1;b | q0;b |
| q1 | q1;b | q2;a |
| q2 | q1;b | q0;b |

Example: Convert Mealy to Moore [NOT CONSISTENT]

*Difficult: Number of states should increase*

Convert the following Mealy Machine to it’s equivalent Moore Machine

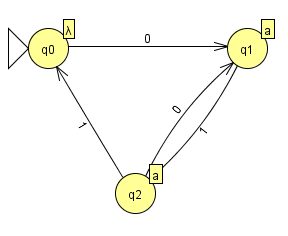
*Over the alphabet**. The machine prints ‘b’ in all other instances*

**A close up of a map

Description automatically generated**

Mealy Machine

|  |  |  |
| --- | --- | --- |
| **State** | **0** | **1** |
| **->**q0 | q1;a | q2;a |
| q1 | q1;b | q2;a |
| q2 | q1;a | q2;b |



To design the machine.

-Draw all the existing states

-Draw the transitions (edges) between states. Only draw the ones which have one associated output.

In this example, the transitions that go from a state to itself are contradicting the output associated with the respective state

i.e. output associated with q1 is a

A close up of a clock

Description automatically generatedi.e. output associated with q2 is a

To design the machine.

-add new states for each transition contradiction

-draw transitions to each new state

To finish the example, we look at q1 and q2. We use these states to draw new transitions. Replicate any transitions from these states.

A picture containing yellow, clock

Description automatically generated

To design the machine.

-Replicate transitions from original states on new states

Moore Machine

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **0** | **1** | **Output (associated with current state)** |
| **->**q0 | q1 | q2 | NULL |
| q1 | q2 | q4 | a |
| q2 | q1 | q3 | a |
| q3 | q1 | q3 | b |
| q4 | q4 | q2 | b |

**Lesson 10**

Finite Automata: Transition Graphs

**Acceptors:** produce binary output, indicating whether the received input is accepted

e.g. deterministic finite automaton (DFA) that detects whether the binary input string contains an even number of 0s.

FA without output

DFA

NFA

NFA

Remember: TGs are how a given automaton is classified as a DFA or NFA. They are simply graphs of transitions between known states and what is required to attain those state transitions

Example: ASS3 Q2

A picture containing clock

Description automatically generated

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

Example: ASS2 Q7

A TG which accepts the language of all strings consisting of at least 4 characters such that the next-to-last letter is identical to the second letter. Use as few states as possible.

*Remember, you can add two or more letters at a time*

L1 = aa[**anything**]aa,ab[**anything**]ba,bb[**anything**]bb,ba[**anything**]ab

A close up of a map

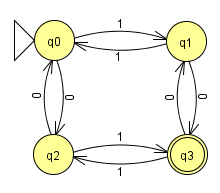
Description automatically generated

**Lesson 11**

Finite Automata: NFA vs DFA

|  |  |  |
| --- | --- | --- |
|  | **DFA** | **NFA** |
|  | **FA:** Finite automata | **TG:** Transition Graphs |
| Type | **Deterministic –** refers to the uniqueness of the computation run | **Non-Deterministic** |
| Restrictions | * each of its transitions is uniquely determined by its source state and input symbol, * reading an input symbol is required for each state transition. | None |
| Explanation | * Can predict next state * One unique next state * No choices or randomness * Simple to design | * Multiple next states * Next state may be chosen at random * All next states may be chosen in parallel |
| # of starts | One | One or more |
| # of states | Finite | Finite |
| # of transitions | Finite | Finite  NULL string (, Empty string) |
| # of input symbols | Finite | Finite |
| # of input strings | ? | Finite |

**DFA: Formal Definition**



It has 5 tuples: (Q, q0, ∑, F, δ)

Q is finite set of states

q0 is the initial state

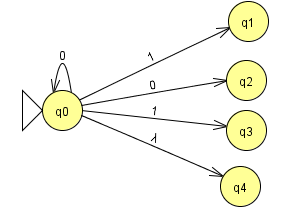
∑ is the input alphabet

F is the set of final states

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

*there are possibilities that a state can go on getting an input*

**NFA: Formal Definition**



It has 5 tuples: (Q, q0, ∑, F, δ)

Q is finite set of states

q0 is the initial state

∑ is the input alphabet

F is the set of final states

δ is input transition function which maps Q×∑ →

*there are possibilities that a state can go on getting an input*

NFA DFA

*Every DFA is an NFA (because Q is part of )*

*But not every NFA is a DFA ((because is NOT part of Q)*

*But there is an equivalent DFA for every NFA*

**How to identify NFA**

* Has (empty string) in one of it’s transitions/edges
* Has (nowhere) as one of it’s transitions/edges
* Has a choice (where one state transitions into more than 2 states)

Example: set of all strings over 0,1 that start with ‘0’

This is an NFA:

input ‘0’: goes to B

input ‘1’: goes to

**Conversion of NFA to DFA**

Example: convert L, the set of all strings over (0,1) that start with ‘0’

to a DFA

*Add a dead state*

A picture containing clock, drawing

Description automatically generated**NFA DFA**

**A close up of a clock

Description automatically generated**

|  |  |  |
| --- | --- | --- |
| **State** | **0** | **1** |
| ->q0 | q1 |  |
| q1 | q1 | q1 |

|  |  |  |
| --- | --- | --- |
| **State** | **0** | **1** |
| ->q0 | q1 | q2 |
| q1 | q1 | q1 |
| q2 | q2 | q2 |

**Minimisation of DFA**

A close up of a clock

Description automatically generated

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

Write down non-final 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 that goes to the final state

**D** goes to the final state, so remove it

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

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

A is like C

**B** is not like to A or C

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

**A close up of a clock

Description automatically generated**

|  |  |  |
| --- | --- | --- |
| State | 0 | 1 |
| ->AC | B | AC |
| B | B | D |
| D | B | E |
| E | B | C |

Example: Are two NFA’s equivalent or not?

Option 1: Minimise both to their minimal DFA’s (minimal DFA’s are unique)

If their minimal DFA’s are equivalent, then the two NFA’s must also be equivalent

Option 2: Use

If accepts any words, the two NFA’s are equivalent.

If accepts any words, the two NFA’s are equivalent.

If X is the empty set, the two NFA’s are equivalent

Before we can try this, we need to understand set theory for FA’s

**Negate FA’s**

Final states are now non-final states

Non-final states are now final states

A close up of a clock

Description automatically generatedA picture containing drawing, clock

Description automatically generated

**Intersection of FA’s**

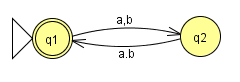
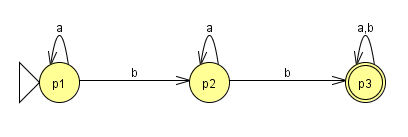
Start with initial states of both FA’s (write both in state)

Create a new state z1 (based on both initial states p1q1)

Create a new state z2 (based on p1q2)

…

Only add the final state where both final states of the FA’s are together (p3q1)



|  |  |  |
| --- | --- | --- |
| State | a | b |
| ->p1q1 =z1 | p1q2=z2 | p2q2=z3 |
| p1q2=z2 | p1q1=z4 | p2q1=z4 |
| p2q2=z3 | p2q1=z4 | p3q1=z5 |
| p2q1=z4 | p2q2=z3 | p3q2=z6 |
| +p3q1=z5 | p3q2=z6 | p3q2=z6 |
| p3q2=z6 | p3q1=z5 | p3q1=z5 |

**Union of FA’s**

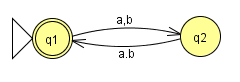
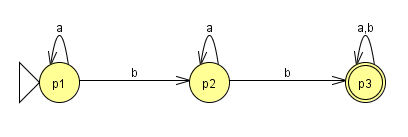
Start with initial states of both FA’s (write both in state)

Create a new state z1 (based on both initial states p1q1)

Create a new state z2 (based on p1q2)

…

???



|  |  |  |
| --- | --- | --- |
| State | a | b |
| ->p1q1 =z1 | p1q2=z2 | p2q2=z3 |
| p1q2=z2 | p1q1=z4 | p2q1=z4 |
| p2q2=z3 | p2q1=z4 | p3q1=z5 |
| p2q1=z4 | p2q2=z3 | p3q2=z6 |
| +p3q1=z5 | p3q2=z6 | p3q2=z6 |
| p3q2=z6 | p3q1=z5 | p3q1=z5 |

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

TODO: finish this

**Exam Curriculum:**

~~Languages: L0 S-A-S-L~~

Regular Expressions: L5

TODO:

~~Provide a regular expression for a required language~~

Analyse a regular expression to determine the language generated.

**Math tools:**

~~Decidability L2~~

~~Recursive Definitions: L3~~ R-C

~~Mathematical Induction: L4~~  C-I-P

~~Pumping Lemma with Length L6~~

**Machines:**

~~FAs (Finite Automata) L8~~

~~TGs (Transition Graphs) L10~~

~~DFAs (Non-deterministic Finite Automata),~~

~~NFAs (Non-deterministic Finite Automata),~~

~~Are two FA’s equivalent or not?~~

~~Convert NFA to DFA L11~~

~~Equivalent NFA L11~~

~~Language Acceptors (Mealy and Moore machines) L9~~

TODO:

~~How to draw each machine~~

~~Determine language being accepted by machine~~

Determine output of Mealy Machine L9

Determine output of Moore Machine L9

~~Convert Moore to Mealy L9~~

~~Convert Mealy to Moore L9~~

Draw Mealy to Moore given the output of a machine

Kleene’s Theorem: TG -> RE, RE -> FA