Group Members

Ogutu, Cindy Atieno – 158842

Mugeni, Amy Zawadi – 167181

Mudibo, Sean Kadida – 168329

Question 1:

Mealy automata represent computational models where the output depends on the machine's present state and the received input; on the other hand, a Moore automaton represents a computational model where the output depends only on the machine's current state.

These are the tuples that denote the Mealy machine:

Q = The complete collection of possible states

$\sum$ = Valid input symbols that form the alphabet

Δ = Contains all possible outputs

∂ = State transition function ($Q \times \sum \rightarrow Q$)

λ = Output mapping ($\sum \times Q \rightarrow \Delta$)

q0 = Initial state

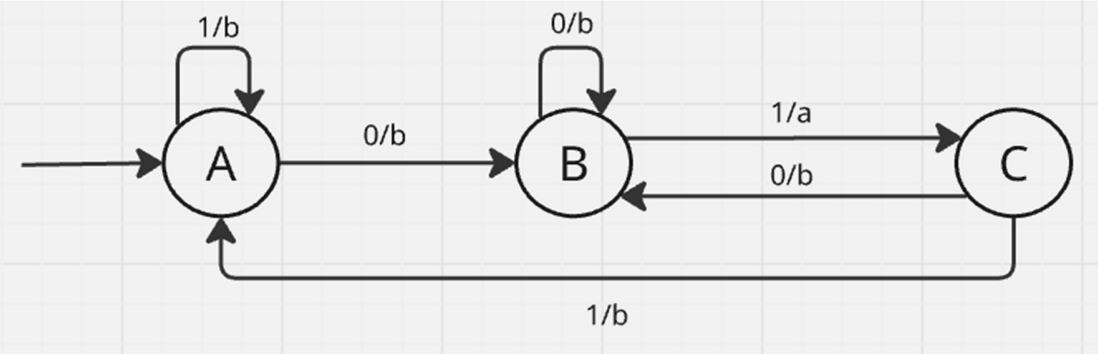
These tuples denote the Moore machine:

All the symbols that denote the Mealy machine apply to the Moore, with the only difference coming in the following tuples:

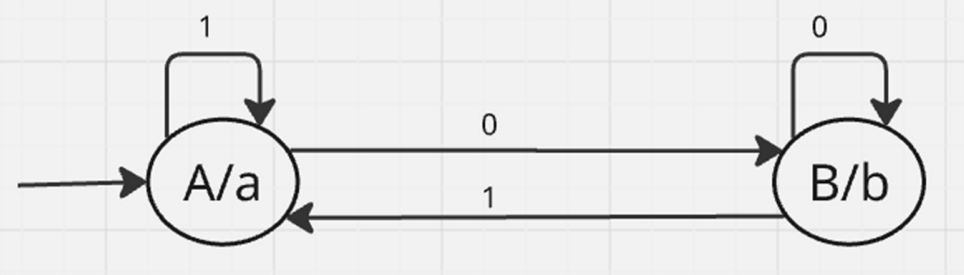
$\partial = (Q \times \sum \rightarrow Q)$

$\lambda = (Q \rightarrow \Delta)$.

The figure below demonstrates the implementation of a Mealy machine. For example, at state A, when an input of 1 is realized, the next state will be A and an output of b will be achieved.



Below is an example of a Moore machine. At state A, the output A will always be achieved, irrespective of the input. At state B, an output of b will always be obtained, whether or not the input is 0 or 1, proving that the output of the Moore machine is completely independent of the input supplied, but fully dependent on the current state only.



Question 2:

Nondeterministic Finite Automata (NFA) - in NFA, for any input data, the state automata can shift to any combination of the states described in the machine hence having a finite number of states.

Deterministic Finite Automata (DFA) - in DFA, for any input data, the state machine can only move to one next state from the current state without accepting any null values.

Key theorems considered

Corollary 1.40 - states that a language is regular if there is a non-deterministic finite automaton (NFA) that acknowledges it. This in turn means that NFAs and DFAs have equal power in realizing and acknowledging regular languages.

Theorem 1.39 - states that if a language is recognized and accepted by an NFA, then it is should also be accepted by a DFA. This theorem implies the use of the subset construction theorem where

the DFA keeps track of all possible states that the NFA could be in by allowing a singular transition for an input to the next state.

Sub Construction algorithm- This is an algorithm that is used to transform a non-deterministic finite automata state automata to a deterministic finite automata state machine. It focuses on the key principle that every NFA has an equivalent DFA in that each state in the DFA is a result of a set in the first NFA state machine. Given the NFA can have null values as well as multiple states for a single input, the DFA simulates this by showing all the possible states the NFA could be in with a single transition from the current to the next state while still maintaining the same constraints.

To demonstrate how the algorithm works, the Key concepts to consider are that with NFA, it allows for ε - transitions (epsilon transitions) that enable multiple transitions of a single input while DFA allows for only one transition for each input to the next state.

Steps to converting NFA to DFA with Subset Construction Algorithm

a) **Create a transition table of the given NFA state machine**
   To transform a Non-deterministic Finite diagram to its identical transition table, we must identify all the states including the start and the accepts states, the input symbols and the transitions between each state. This will be represented in a tabular structure where the columns represent the present states, and the rows represent the input signals. The rows and columns populating the table represent the next states.
b) **Identify the NFA start state.**
   It is relevant to identify the start state of the NFA as it serves as the same start state for the DFA. This state is known as the Epsilon closure as it serves as a set where all the states can be attained from the beginning by following the epsilon transitions.
c) **Iteratively construct DFA states and transitions**
   While constructing the transition table, there will be some unmarked DFA states making it difficult to compute the next states. To go about this, for very unmarked state, mark it with a symbol (N) and for each input (α) perform a union ($\cup$) between all the states reachable from (N) with the input symbol (α). The (T) result of that union will form the next state for the DFA.

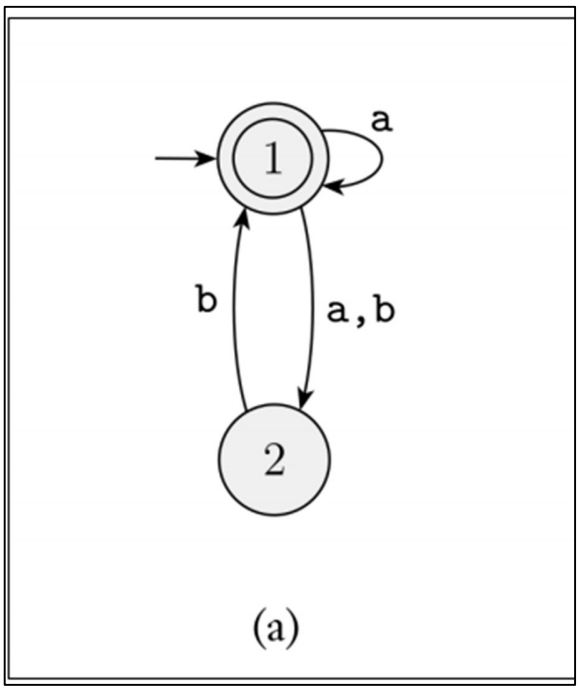
d) Determine the DFA's final states

To determine the DFA's final states, these will be states (one or multiple) that contain at least one final state from the NFA being converted.

e) DFA minimisation if necessary

This is the process of coming up with a minimal version of the DFA, by using the least number of states possible while still following the same transition rules. This is to make the diagram more efficient for use and interpretation.

Solving Question (a) with Sub Construction theorem



i) Step 1, Transitional table for the NFA above

States/Input	a	b
1	{1,2}	{2}
2	∅	{1}

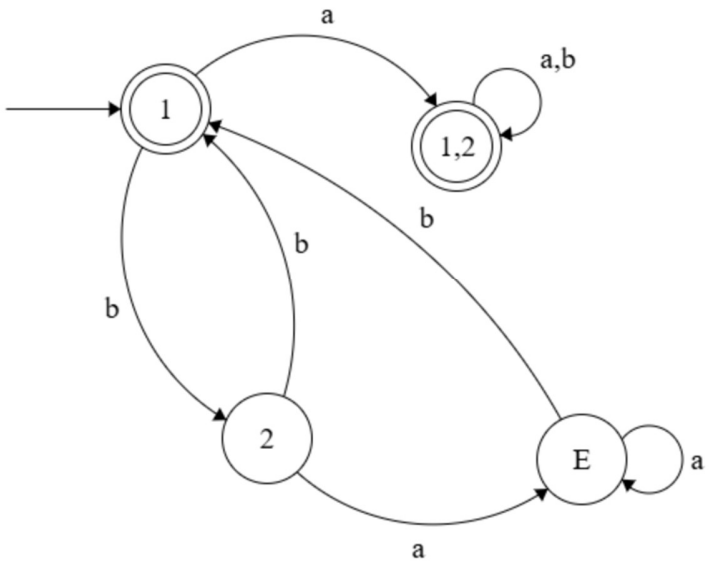
ii) Step 2, Identify the start state of the NFA

States/Input	a	b
1	**{1,2}**	**{2}**
2	∅	**{1}**

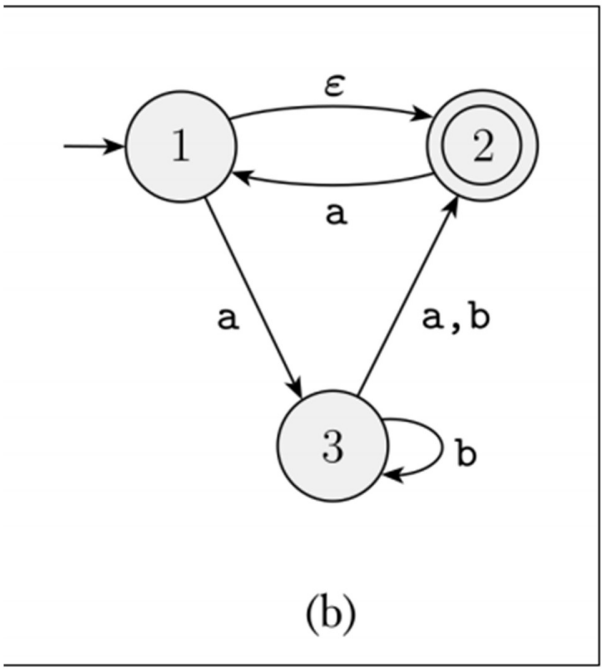
iii) Step 3, Iteratively construct DFA state transitions

States/ Input	a	b
1	{1,2}	{2}
1,2	{1,2}	{1,2}
2	E	{1}
E	E	{1}

iv) Step 4, DFA Final states



Solving Question b with Sub-Construction Algorithm



i) Step 1, Transitional table for the NFA above

States/ Inputs	a	b
1	3	∅
3	2	{2,3}
2	1	∅

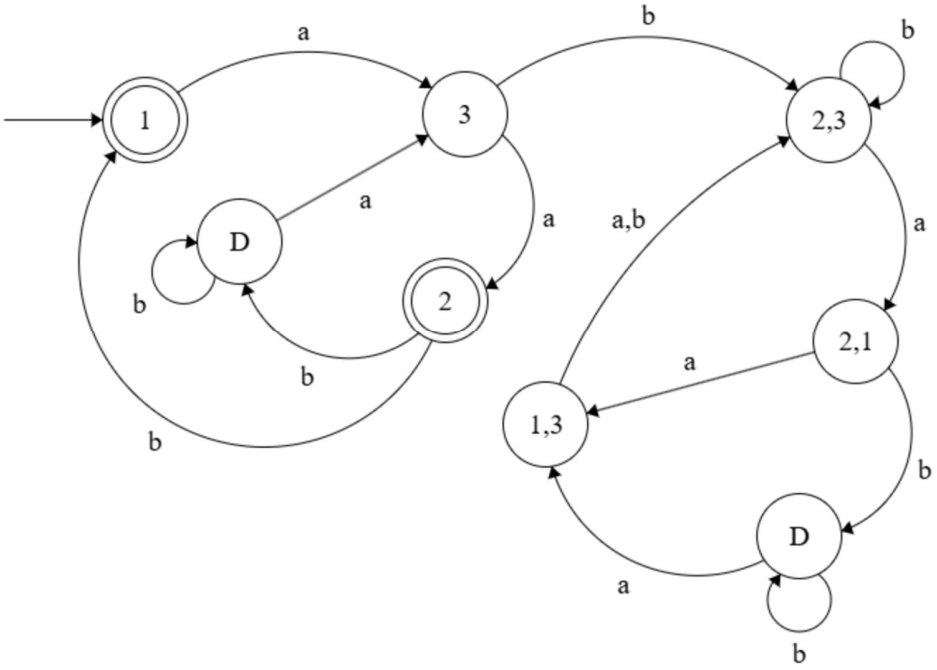
ii) Step 2, Identify the start state of the NFA

States/ Inputs	a	b
1	3	∅
3	2	{2,3}
2	1	∅

iii) Step 3, Iteratively construct DFA state transitions

States/ Inputs	a	b
1	3	Ø
Ø	3	Ø
3	2	{2,3}
2	1	Ø
Ø	1	Ø
{2,3}	{2,1}	{2,3}
{2,1}	{1,3}	Ø
Ø	{1,3}	Ø
{1,3}	{2,3}	{2,3}

iv) Step 4, DFA Final states

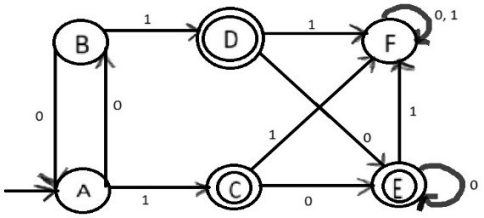


Question 3:

- **DFA minimisation** is the process by which the minimal version of a Deterministic Finite Automaton (a Finite Automaton without an output) is obtained. It comprises the minimum number of states possible. A DFA with fewer states is considered more efficient.
- The main idea behind DFA minimisation is to put all states that are equal together. Take two states, A and B. They are considered equal if, upon receiving any string input, say Z, they either both end up in a final state or both do not end up in a final state. There are types of equivalence based on the length of the input string Z:
  - If the length of string Z is 0, states A and B are said to be **0-equivalent.**
  - If the length of string X is 1, states A and B are said to be **1-equivalent**
  - Hence, in general, if the length of string Z is *n,* states A and B are **n-equivalent.**
- The conversion of Non-Deterministic Finite Automata to an equivalent Deterministic Finite Automata can be achieved using the **subset construction method.** This method ensures that the resulting DFA accepts the same language as the original NFA.
- The following steps outline how a language model can be minimized using the subset construction method to first get an equivalent DFA.
  1. Start by understanding the NFA's characteristics. An NFA allows for more than one next state for a given state and input symbol. For example, a state 'b' on an input '0' might go to both 'c' and 'd' at the same time. The transition function maps a state and an input symbol to a set of possible next states, rather than a single unique state. In an NFA, it is allowed to leave a state without mentioning the path it takes on a particular input; it means it goes nowhere unlike DFAs where every state must have a specific transition for all inputs.
  2. Then go ahead with the subset construction process. The main idea is to simulate the NFA's non-determinism by **tracking all possible states** the NFA could be all at once. Each state in the new DFA will then represent a set of states from the original NFA. Begin with the initial state of the new DFA which will be the sent containing only the initial state of the NFA.
  3. The process of minimising a DFA involves:

A. **Draw the state transition table** for the given DFA. This table defines how each state transitions on receiving different inputs.
B. **Separate all non-final states** into one set and all **final states** into another set. Example, of 'E' is the only final state and 'A,B,C,D' are non-final, then your sets will be {A, B, C, D} and {E}. If the states are multiple, they are grouped together.
C. For each subsequent equivalence level, you examine the steps formed in the previous step. For states within the same set, check if they are equivalent by observing their transitions for all input symbols. If their transitions lead to states that fall into different sets from the previous equivalence level, then the states are **not equivalent** and must be separated into new sets. The process continues iteratively.
D. The process stops when the sets of equivalent states at the current level **happen to be the same as** the sets from the previous level.
E. The final set of states represent the new, combined states for **minimised DFA.** A new transition table and state diagram are then constructed using these combined states.

- DFA minimisation is **significant to complexity** primarily in terms of efficiency and amount of space resources consumed. Finite State Machines which include DFAs and NFAs, are the simplest models of computation and they have minimal memory as they can only recall their current state and are unable to store or count strings. Hence, a DFA with fewer stats directly translates to a more **space-efficient algorithm** as it implies less 'memory' overhead. The computation becomes more streamlined and easier for languages that do not require remembering more than the current state.
- Below is an example to justify the complexity argument:

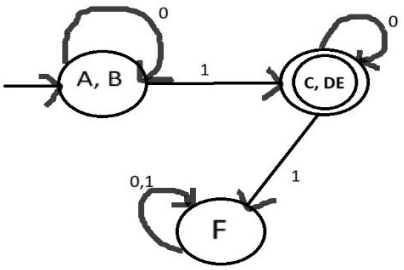


	0	1
A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

0- Equivalence - {A, B, F} {C, D, E}

1- Equivalence - {A,B} {F} {C,D,E}

2- Equivalence - {A,B} {F} {C,D,E}



	0	1
{A, B}	{A,B}	{C,D,E}
{F}	{F}	{F}
{C,D,E}	{C,D,E}	{F}

The example provided shows a DFA with five states (A,B,C,D,E) being minimised to a DFA with four states. Both DFAs are performing the same task but the four-state DFA is more efficient. The reduction in the number of states directly relates to the **space complexity** of the automaton. By making use of less states, the machine utilizes less memory to represent its ongoing transitions. Hence the minimal DFA is an optimal solution in terms of computational resources, especially given that the finite automata has limited memory capacity.

REFERENCES

GeeksforGeeks. (2023, January 20). *Conversion from NFA to DFA*. GeeksforGeeks. https://www.geeksforgeeks.org/conversion-from-nfa-to-dfa/

TutorialsPoint. (2025, March 25). *NDFA to DFA conversion*. https://www.tutorialspoint.com/automata_theory/ndfa_to_dfa_conversion.htm

Sajeeb. (2025, May 24). The subset construction algorithm: NFA/E-NFA to DFA. *Medium*. https://medium.com/@mmksajeeb/the-subset-construction-algorithm-nfa-%CE%B5-nfa-to-dfa-adf46dba31e3

Mealy and Moore Machines in TOC. (2016, June 5). GeeksforGeeks. https://www.geeksforgeeks.org/mealy-and-moore-machines-in-toc/