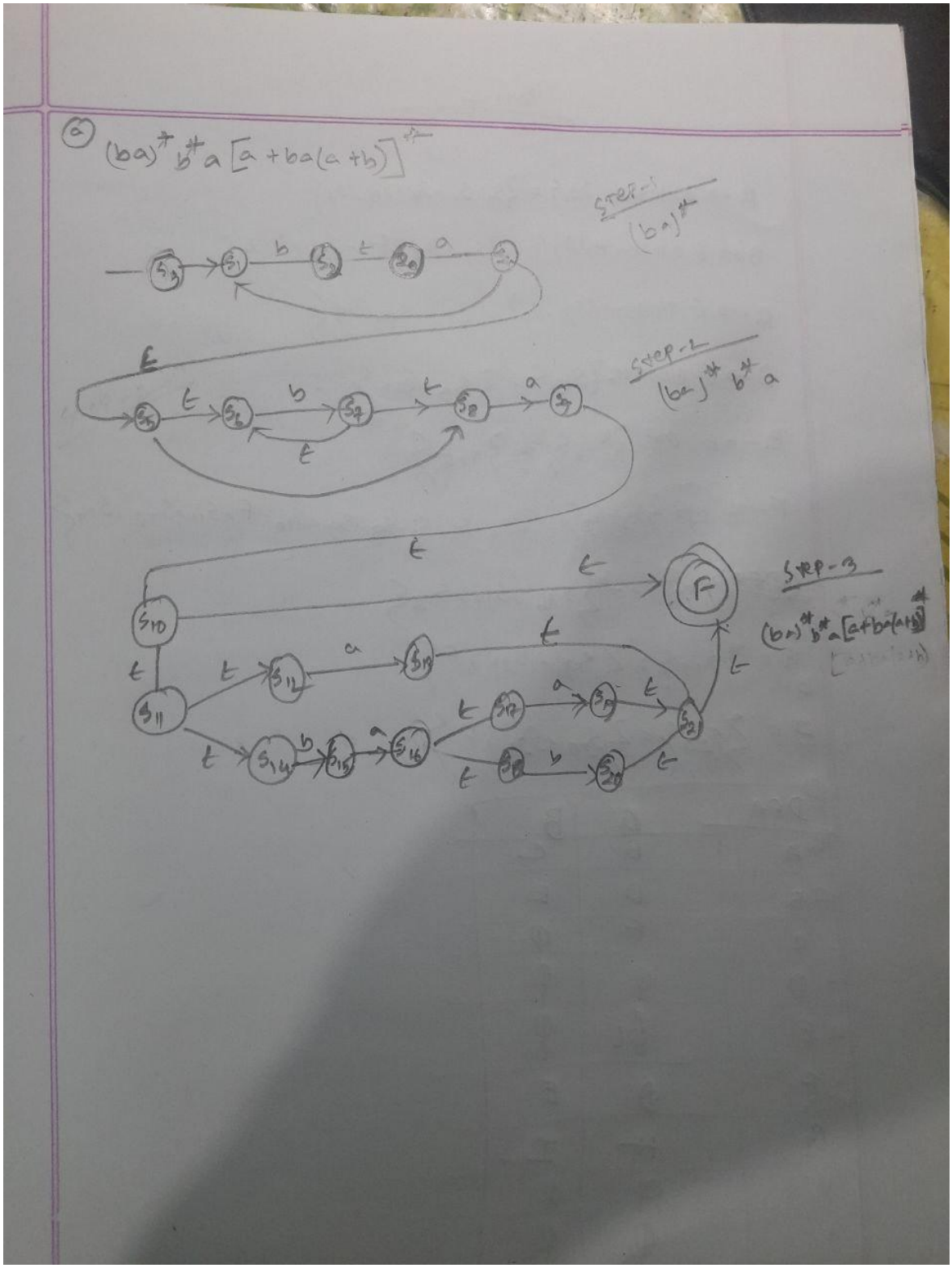
MD Oakil Sarker
21-45284-2
Compiler Design[B]

(a) $(ba)^* b^* a [a + ba(a+b)]^*$



Ques-2

A → ε closure $\{S_0\} = \{S_0, S_1, S_4, S_7, S_8\}$

B → ε closure $\{S_3\} = \{S_3, S_{10}, F, S_{11}, S_{12}\}$

C → ε closure $\{S_2, S_7\} = \{S_2, S_7, S_8\}$

D → ε closure $\{S_{13}\} = \{S_{13}, S_{21}\}$

E → ε of $\{S_{15}\} = \{S_{15}\}$

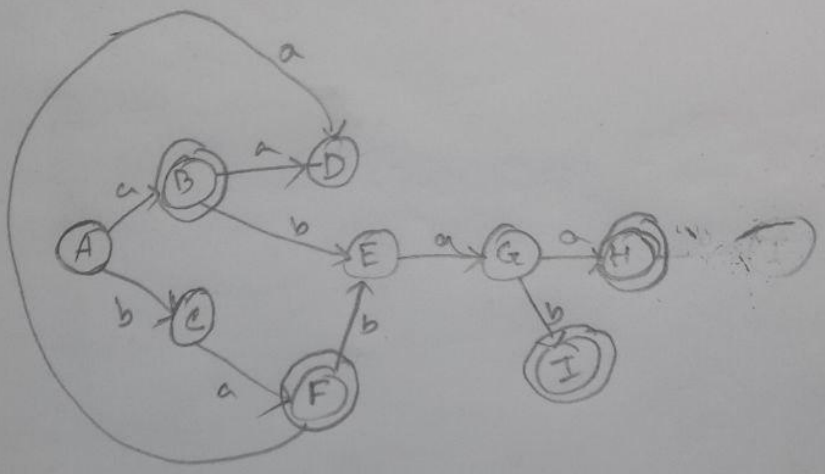
F → ε of $\{S_3, S_7\} = \{S_4, S_5, S_6, S_8, S_9, S_{10}, F, S_{11}, S_{12}, S_{14}\}$

G → ε of $\{S_{16}\} = \{S_{16}, S_{19}, S_{18}\}$

H = $\{S_{19}\} = \{S_{19}, S_{21}, F\}$

I = $\{S_{20}\} = \{S_{20}, S_{21}, F\}$

DFA	a	b
A	B	C
B	D	E
C	F	ϕ
D	ϕ	ϕ
E	G	ϕ
F	D	E
G	H	I
H	ϕ	ϕ
I	ϕ	ϕ


a
a
a
B
D
A
b
E
a
G
a
H
b
C
b
a
F
I

DFA IMPLEMENTATION IN C++

```
#include <iostream>
#include <string>
using namespace std;

class DFA {
private:
    enum State { q0, q1, q2, q3, q4, q5, qd };
    State currentState;

public:
    DFA() {
        currentState = q0;
    }

    void reset() {
        currentState = q0;
    }

    bool process(const string& input) {
        reset();

        for (char c : input) {
            if (c != 'a' && c != 'b') {
                return false; // invalid symbol
            }

            switch (currentState) {

            case q0:
                if (c == 'b') currentState = q1;
                else currentState = q3;
                break;

            case q1:
                if (c == 'a') currentState = q0;
                else currentState = q2;
                break;

            case q2:
                if (c == 'b') currentState = q2;
```

```
            else currentState = q3;
            break;

        case q3: // accepting
            if (c == 'a') currentState = q3;
            else currentState = q4;
            break;

        case q4:
            if (c == 'a') currentState = q5;
            else currentState = qd;
            break;

        case q5:
            currentState = q3;
            break;

        case qd:
            currentState = qd;
            break;
        }
    }

    return currentState == q3;
  }
};

int main() {
  DFA dfa;
  string input;

  cout << "Enter string: ";
  cin >> input;

  if (dfa.process(input))
    cout << "String ACCEPTED by DFA\n";
  else
    cout << "String REJECTED by DFA\n";

  return 0;
}
```

Documentation

Regular Expression:
(ba)* b* a [a + ba(a+b)]*

1. Overview
This document explains the implementation of the given regular expression using a Deterministic Finite Automaton (DFA) and its simulation in C++. The DFA reads input symbols one by one and changes states according to transition rules. A string is accepted if the DFA ends in an accepting state after processing the entire input.

2. Alphabet
The input alphabet is Σ = {a, b}. Any symbol outside this alphabet is rejected.

3. DFA States
q0: Start state, implements (ba)*
q1: Intermediate state after
reading 'b' q2: Implements b*
q3: Accepting state after
mandatory 'a' q4: After reading
'b' in [a + ba(a+b)]* q5: After
reading 'ba' in [a + ba(a+b)]* qd:
Dead (trap) state

4. Implementation of Each Part

(a) (ba)*
The DFA alternates between q0 and q1 on input 'b' and 'a' respectively. This allows zero or more repetitions of the string "ba".

(b) b*
State q2 represents zero or more 'b'. The DFA remains in q2 while reading 'b' and exits when 'a' is encountered.

(c) Mandatory a
After (ba)* and b*, exactly one 'a' is required. The DFA moves to q3, which is the accepting state.

(d) [a + ba(a+b)]*
From q3, the DFA can read 'a' repeatedly or follow the sequence 'b' $\rightarrow$ 'a' $\rightarrow$ ('a' or 'b'), which represents the patterns baa or bab. This is handled using states q4 and q5.

5. Acceptance Condition
A string is accepted if the DFA ends in state q3 after processing the complete input

string.

6. Conclusion

The DFA correctly models the given regular expression by mapping each component to specific

states and transitions. The C++ implementation simulates the DFA efficiently and accurately determines whether an input string belongs to the language