Karachi Institute of Engineering & Technology (KIET)
CoCIS Department

Design and Implementation of Finite Automata Based Validation System

Theory of Automata Project

Muhammad Umair
Roll No: 66144

Course: Theory of Automata
CID: 118691
Instructor: Miss Misbah Anwer

Contents

Abstract

This report present the developement of a validation system using Deterministic Finite Automata (DFA) principals. The system check inputs such as passwords, URLs, email address, and phone numbers. The approach demostrates how automata theory can be applyed to practical problem in software developement. The report include background, related work, methodology, experimental result, conclusion, and refrences.

1 Introduction

1.1 Background

Finite Automata is foundational model in theoretical computer science used for pattern recognization and language parsing. They form the basic of regular languages and is widely teached in Theory of Automata courses. A DFA process a sequence of input symbol and determine whether the input belong to a defined language.

1.2 Motivation

With the increasing need for secure and correct data validation in software system, automata models offers a systematic and determinstic method for validating structured strings. For example, ensuring strong password and valid URLs improve system security and user experiance.

1.3 Significance

By building practical validator using DFA, this project bridge theoretical concept with real world applications. The system is design to be both educational and usefull in real software scenario and learning purpose.

2 Related Work

Several studies has explored automated validation techniques using formal languages and pattern recognization:

Fandino-Mesa et al. describe the use of regular grammer to validate email address, showing how formal language theory applys directly to input validation problem in software systems. They build and tests finite automata for email syntax based on recomended standards [1].

Password strength reserach highlight the need for robust validation system. Studies on password complexity emphasise factor such as character diversity and lenght as determinant of strength [2].

Studies on machine learning model for password strength reveals alternative approach like Markov models and neural network for estimating password security, ilustrating the broader context of validation beyond simple rule based system [3].

Further work on regex analysis underscrores potential vulnerabilties in pattern matching system and the importance of efficent pattern validation method [4].

Research on formal grammer analysis also demostrates the value of regular expressions and grammer in syntactic validation task such as email parsing and lexical analysis process [5].

3 Methodology

The project implement four validator using DFA-like logic and structure:

3.1 Password Strength Checker

A DFA track whether uppercase, lowercase, numeric, and symbolic character have been seen or not. The input is accept only when all condition is met correctly.

3.2 URL Validator

The URL DFA process protocol, domain, top-level domain (TLD), and optional path. It transit systematically through protocol recognization and domain structure rule.

3.3 Email Validator

The email DFA check the presence of a username, an '@' symbol, a domain, and a valid extention. The transition ensure correct sequencing of this component.

3.4 Phone Number Validator

This DFA check phone number format such as international and local standard, ensuring correct prefix and digit count.

Each validator read input character by character and update state accordingly, following determinstic transition function.

4 Results & Experiments

The system was test using a range of valid and invalid example for evaluation:

4.1 Password Strength Validator

Implementation Code

The following code implements a DFA-based password strength checker that ensures the presence of uppercase, lowercase, numeric, and special characters.

```
def check_password(password):
    has_upper = False
    has_lower = False
    has_digit = False
    has_symbol = False

    for ch in password:
        if ch.isupper():
            has_upper = True
        elif ch.islower():
            has_lower = True
        elif ch.isdigit():
            has_digit = True
        else:
            has_symbol = True

    if has_upper and has_lower and has_digit and has_symbol:
        return True
    return False
```

Listing 4.1: Password Strength Validator Code

Experimental Output

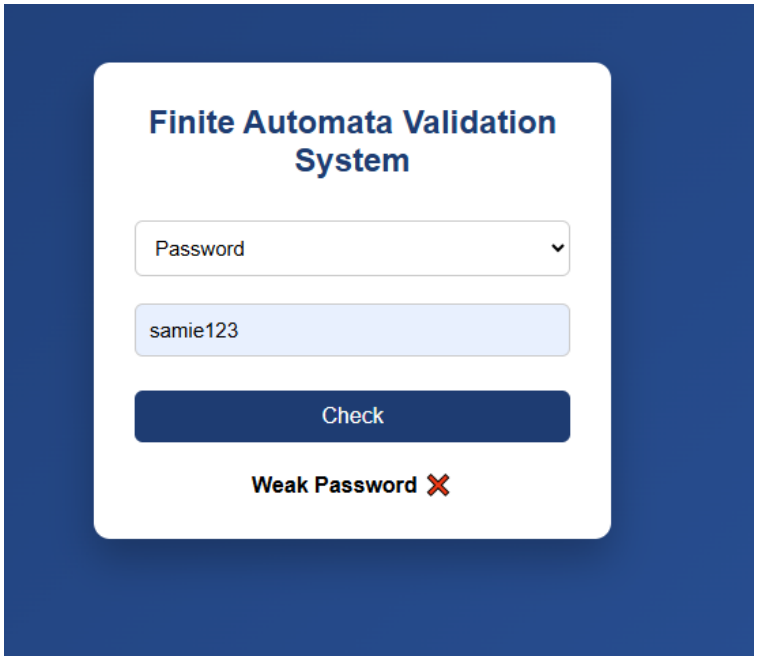

Figure 4.1: Password Validation Result

DFA States

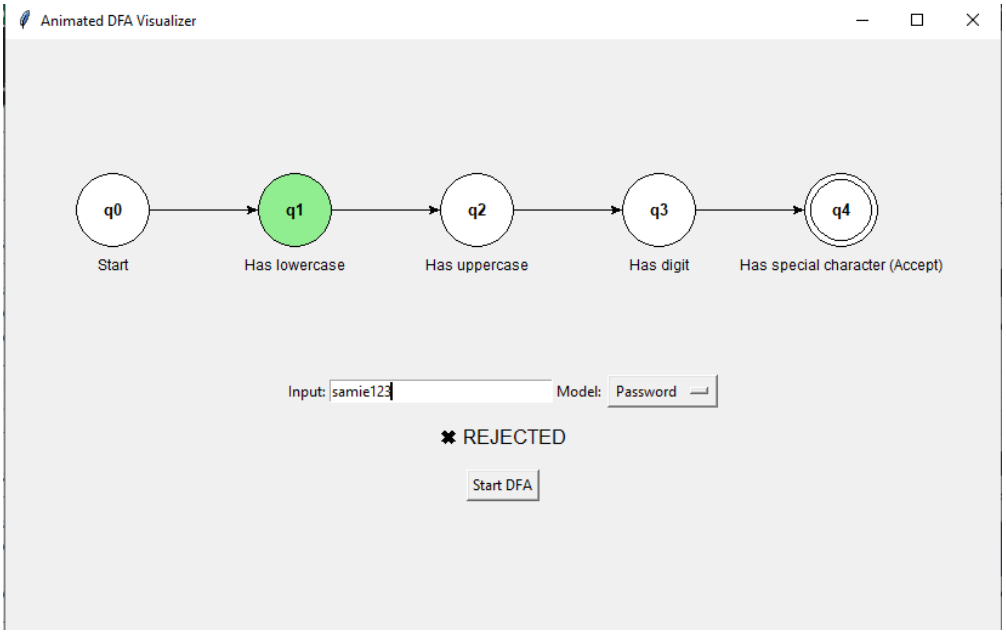

Figure 4.2: Password Validation Result

4.2 URL Validator

Implementation Code

This validator checks whether the given URL follows a valid protocol and domain structure.

```
def check_url(url):
    state = 0
    i = 0

    while i < len(url):
        ch = url[i]

        if state == 0 and url.startswith("http://", i):
            state = 1
            i += 7
            continue
        elif state == 0 and url.startswith("https://", i):
            state = 1
            i += 8
            continue

        elif state == 1 and ch.isalnum():
            state = 2

        elif state == 2 and ch.isalnum():
            pass
        elif state == 2 and ch == '.':
            state = 3

        elif state == 3 and ch.isalpha():
            state = 4

        elif state == 4 and ch.isalpha():
            pass
        elif state == 4 and ch == '/':
            state = 5

        elif state == 5:
            pass
        else:
            return False

        i += 1

    return state in [4, 5]
```

Listing 4.2: URL Validator Code

Experimental Output

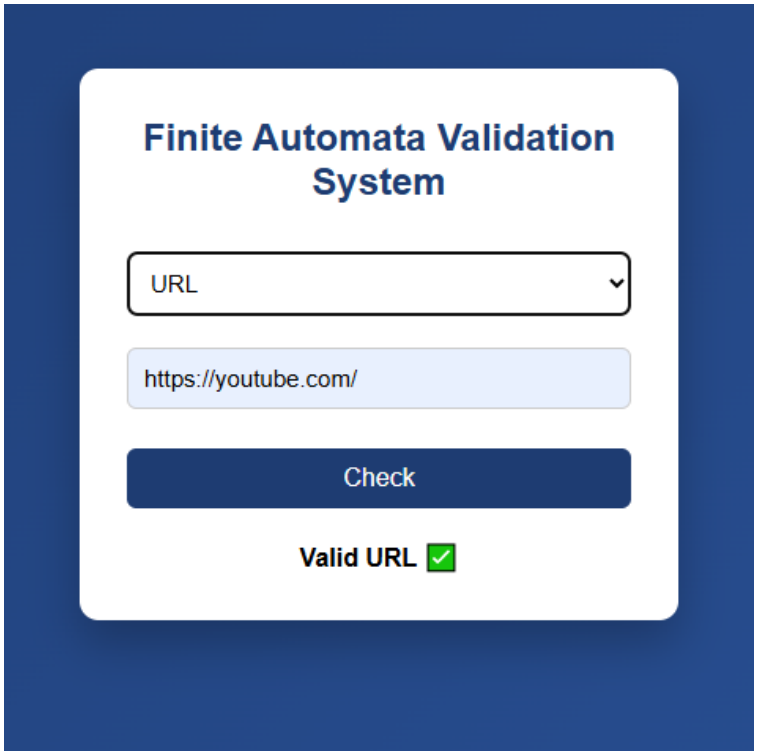

Figure 4.3: URL Validation Result

DFA States

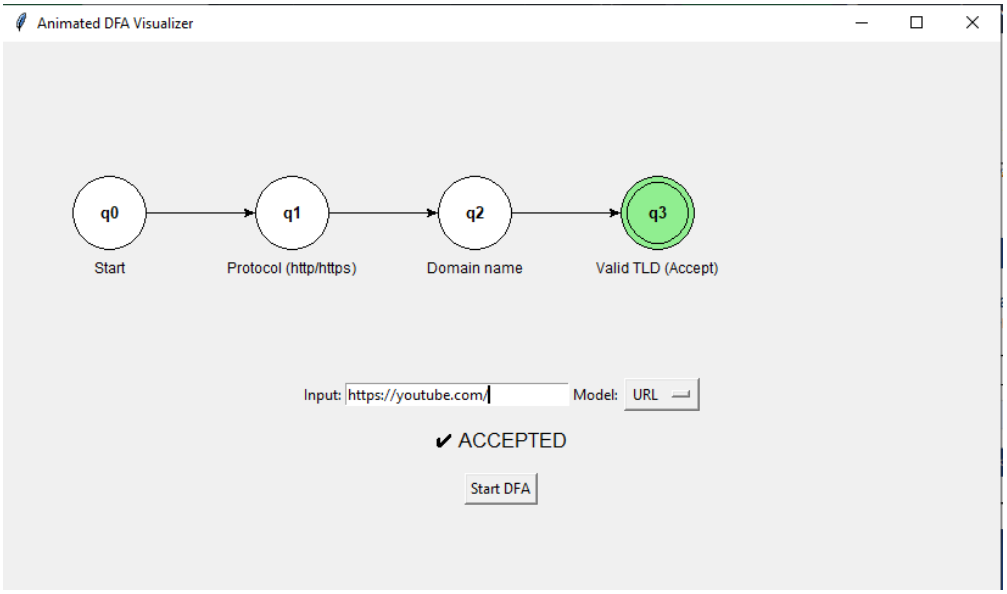

Figure 4.4: URL Validation Result

4.3 Email Validator

Implementation Code

The email validator ensures the correct sequence of username, '@' symbol, domain name, and extension.

```
def check_email(email):
    state = 0

    for ch in email:
        if state == 0:
            if ch.isalnum():
                state = 1
            else:
                return False

        elif state == 1:
            if ch.isalnum():
                pass
            elif ch == '@':
                state = 2
            else:
                return False

        elif state == 2:
            if ch.isalnum():
                state = 3
            else:
                return False

        elif state == 3:
            if ch.isalnum():
                pass
            elif ch == '.':
                state = 4
            else:
                return False

        elif state == 4:
            if ch.isalpha():
                state = 5
            else:
                return False

        elif state == 5:
            if ch.isalpha():
                pass
```

```
            else:
                return False

    return state == 5
```

Listing 4.3: Email Validator Code

Experimental Output

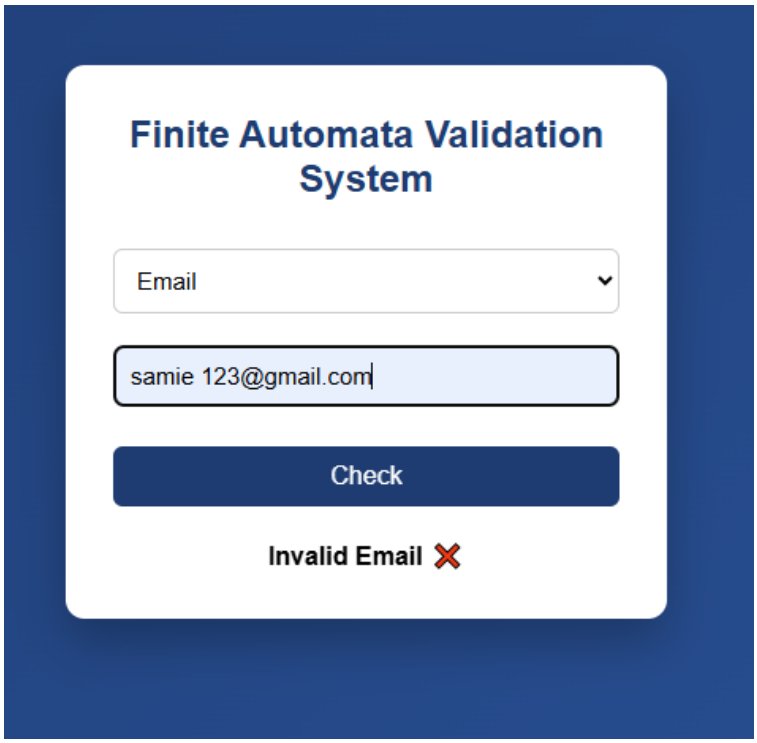

Figure 4.5: Email Validation Result

DFA States

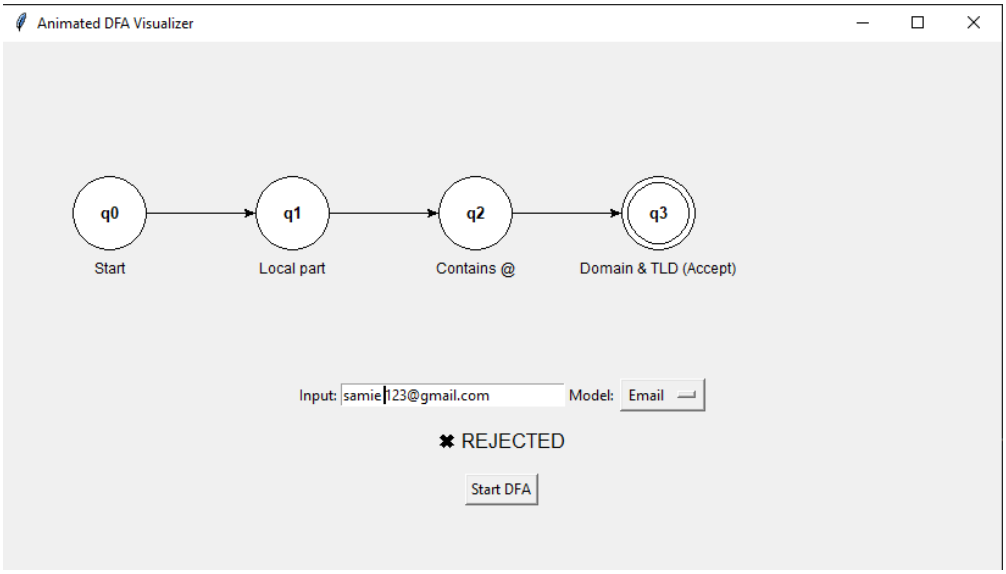

Figure 4.6: Email Validation Result

4.4 Phone Number Validator

Implementation Code

This validator checks phone numbers for valid prefixes and correct digit length.

```
def check_phone(phone):
    if phone.startswith("+92") and len(phone) == 13:
        return phone[3:].isdigit()

    if phone.startswith("03") and len(phone) == 11:
        return phone.isdigit()

    return False
```

Listing 4.4: Phone Number Validator Code

Experimental Output

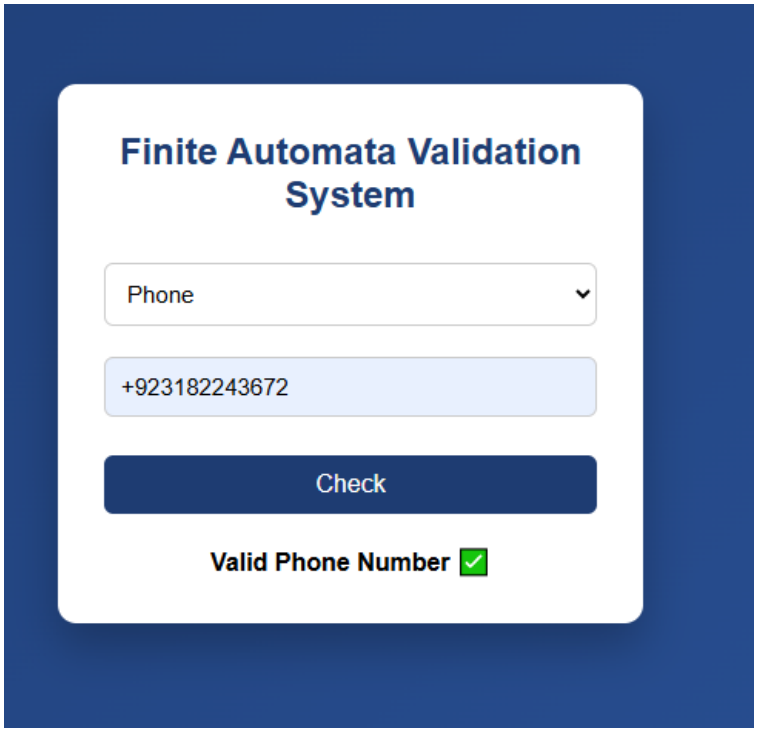

Figure 4.7: Phone Number Validation Result

DFA States

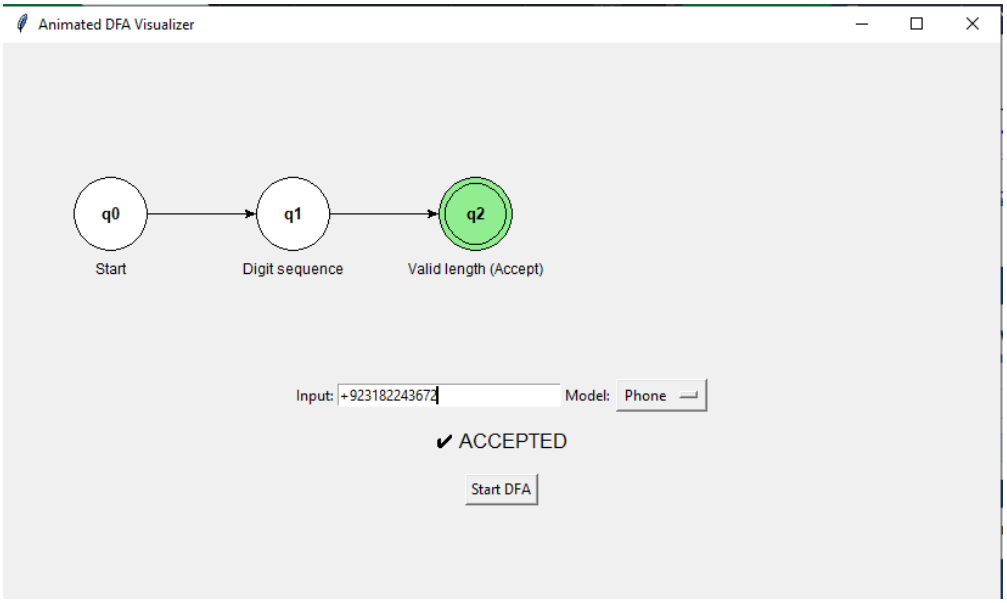

Figure 4.8: Phone Number Validation Result

4.5 Summary of Results

The validator consistantly accepted valid input and rejected incorrect format, demostrating the effectiveness of DFA-based logic system.

5 Conclusion

This project demostrates how deterministic finite automata can be used to validate structured string pattern. The approach model real world input validation problem within a formal theoretical framework. The result confirm that DFA principals can provide reliable and determinstic outcome for common validation task.

6 References

Here are the link to the research used in this project:

1. C. A. Fandino-Mesa, M. Suarez-Baron, and C. Jaramillo-Acevedo, "Application of Regular Grammar in the Syntactic Analysis of Email Addresses," *Ingeniería*, 2023. Available at: https://revistas.udistrital.edu.co/index.php/reving/article/view/20626

2. K. Chanda, "Password Security: An Analysis of Password Strengths and Vulnerabilities," *International Journal of Computer Network and Information Security*, 2016. Available at: https://www.mecs-press.org/ijcnis/ijcnis-v8-n7/v8n7-4.html

3. M. Taneski et al., *Strength Analysis of Real-Life Passwords Using Markov Models*, IJARIIE, 2025. Available at: https://ijariie.com/AdminUploadPdf/SURVEY_ON_PASSWORD_STRENGTH_ANALYZER_USING_LSTM_AND_CNN_ijariie26050.pdf

4. J. Kirrage, A. Rathnayake, and H. Thielecke, "Static Analysis for Regular Expression Denial-of-Service Attacks," 2013. Available at: https://arxiv.org/abs/1301.0849

5. Applied Information Technology and Computer Science, *Implementation of Username and Password Strength Validation*, 2025. https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/download/16539/6310