Formal Language Simulator – User Manual

Overview

This program is a **pattern searching simulator**. It helps you search for patterns in **text or DNA sequences** using concepts from computer science.

You **do not need to know automata theory** to use this program. It will:
1. Check if a string exactly matches a pattern.
2. Convert the pattern into **automata** (NFA and DFA) for visualization and faster checking.
3. Search for patterns in **DNA sequences**, allowing small differences (like mutations).
4. Show the steps the program takes in an easy-to-read format.

How the Program Works (Simplified)

1. Pattern Input (Regex)

- You type a **pattern** you want to search for.
- Example: `ab*` → matches `a`, `ab`, `abb`, `abbb`, etc.
- The program builds a **map of states and transitions** from this pattern.

2. Exact Match (NFA & DFA)

- The program takes your test string and checks if it matches the pattern:
- **NFA (Non-deterministic Finite Automaton):** Explores all possible paths at once.
- **DFA (Deterministic Finite Automaton):** Follows only one strict path.
- If the string can reach the **final state**, it is a match.

3. Approximate DNA Matching

- Enter a DNA sequence using only `A`, `C`, `G`, `T`.
- The program checks if your pattern appears in the DNA, **allowing up to 1 difference** (mutation).
- Useful for real biological sequences where errors or mutations may exist.

4. Transition Visualization

- The program prints all **states and transitions** of both NFA and DFA.
- Example:

```
0 --a--> 1
1 --b--> 2
Start state: 0
Final states: 2
```

- This helps you see **how the program processes each string**.

Step-by-Step Usage

1. **Run the program**
   Compile the code (example using g++):
   ``` g++ -o simulator simulator.cpp ./simulator # Linux / Mac simulator.exe # Windows

```
   Or run in **VS Code terminal** or **IDE** (console will stay open).

2. **Enter a regex pattern**
   Example: `ab*`

3. **Enter a string to test exact match**
   Example: `abb`

4. **Enter a DNA sequence for approximate matching**
   Example: `ACGTACG`

5. **Read results**
   - The program will tell you if the string matches the pattern exactly.
   - It will also tell you if the DNA sequence matches approximately.
   - NFA and DFA transitions are printed for visualization.

6. **Exit the program**
   - The program will prompt:
   ```
   Press Enter to exit...
   ```

Example Run
```

   Enter a regex pattern (simple characters supported, e.g., ab*): *ab* NFA Transitions: 0 --a--> 1 1 --b--> 2 Start state: 0 Final states: 2

DFA Transitions: 0 --a--> 1 1 --b--> 2 Start state: 0 Final states: 2

Enter a string to test exact match: abb Match found using NFA! Match found using DFA!

Enter a DNA sequence for approximate matching: abbc Approximate match found with at most 1 error(s)!

Press Enter to exit... ```

## Tips for Users

- **Regex patterns supported:**
- Single characters: `a`, `b`, etc.
- Concatenation: `ab` → matches exactly `ab`
```

- Kleene star `*`: `ab*` → matches `a`, `ab`, `abb`, ...
- **DNA sequences:** Only use `A`, `C`, `G`, `T`. Uppercase recommended.
- **Console stays open:** Always run from a **terminal** or **IDE**. The program pauses at the end to let you read results.
- **Visualization:** NFA shows **flexible paths**, DFA shows **strict paths**. Use it to understand how the program processes patterns.