Formal Language Simulator – User Manual

Overview

The Formal Language Simulator is a console-based educational tool designed to demonstrate how different classes of formal languages are recognized using different computational models. The program integrates finite automata, approximate pattern matching, and pushdown automata into a single system.

It is intended for students studying formal languages, automata theory, compilers, and bioinformatics. No prior in-depth knowledge of automata theory is required to use the program.

The simulator supports: - Regular language recognition using NFA and DFA - NFA to DFA conversion (subset construction) - Approximate pattern matching for biological sequences (DNA) - Context-free language recognition using a Pushdown Automaton (PDA)

How the Program Works (Simplified)

The program is divided into four main functional components, each corresponding to a different class of formal language or recognition technique.

1. Pattern Input (Regular Language)

You first enter a pattern consisting of literal characters. Each character is treated as a symbol in a regular language.

Important Note: The program supports literal concatenation only. Special regex operators such as * or | are treated as normal characters.

Example Pattern

-ab

This pattern represents the regular language containing exactly the string ab.

From this pattern, the program: - Constructs an NFA (Non-deterministic Finite Automaton) - Converts the NFA into an equivalent DFA (Deterministic Finite Automaton)

2. Exact Match Using NFA and DFA

After building the automata, the program asks for a test string.

Both the NFA and DFA simulate the input string: - The NFA explores all possible transitions simultaneously - The DFA follows a single deterministic transition path

If the simulation ends in a final state, the string is accepted.

Sample Input

Pattern: ab
Test string: ab

Output

NFA ACCEPT
DFA ACCEPT

3. Approximate Pattern Matching (DNA Sequence Analysis)

The simulator supports approximate matching using an edit-distance–based dynamic programming algorithm.

This feature models real-world DNA sequence analysis, where small differences (mutations, insertions, deletions) are common.

The program checks whether the pattern appears as a substring in the given DNA sequence within a user-defined maximum number of errors.

Sample Input

Pattern: ACG
DNA sequence: TACGGA
Maximum errors: 1

Output

Approximate match found

This means the pattern ACG appears in the DNA sequence with at most one difference.

4. Transition Visualization (NFA and DFA)

For transparency and learning purposes, the program prints the transition tables for both automata.

Example NFA Output

0 --a--> 1
1 --b--> 2
Start: 0
Final: 2

Example DFA Output

0 --a--> 1
1 --b--> 2
Start: 0
Final: 2

These outputs help illustrate how strings are processed step by step by each automaton.

---

5. Context-Free Language Recognition (Pushdown Automaton)

To demonstrate the limitations of regular languages, the program includes a Pushdown Automaton (PDA).

The PDA recognizes the classic context-free language:

$L = \{ a^n b^n \mid n \geq 1 \}$

This language cannot be recognized by finite automata alone and requires a stack, which is provided by the PDA.

PDA Behavior

- Push one symbol onto the stack for each a
- Pop one symbol from the stack for each b
- Accept if the stack is empty and the input is fully consumed

Sample Input

aabb

Output

PDA ACCEPT (a^n b^n)

Rejected Example

aabbb

Output:

PDA REJECT

Step-by-Step Usage

1. Compile the program

```
g++ -std=c++17 searchsystem.cpp -static-libgcc -static-libstdc++ -O2 -o formal_sim.exe
```

2. Run the program
3. Enter a pattern (literal concatenation only)
4. Enter a string for exact matching
5. Enter a DNA sequence and maximum allowed errors
6. Enter a string for PDA testing (a^n b^n form)
7. Review the results and transition tables

Sample Complete Run

```
Enter regex: ab
Enter string for exact match: ab
NFA ACCEPT
DFA ACCEPT

Enter DNA sequence: xabx
Enter max errors: 1
Approximate match found

Enter string for PDA test: aaabbb
PDA ACCEPT (a^n b^n)
```

Tips for Users

- Use uppercase A, C, G, T for DNA input
- Avoid spaces in inputs (the program reads tokens)
- Remember that regex operators like * are treated as literal characters
- Use the PDA section to understand why some languages are not regular

Summary

This simulator demonstrates: - Regular language recognition using NFA and DFA - Performance-oriented conversion from NFA to DFA - Approximate matching for biological sequences - Context-free language recognition using Pushdown Automata

It clearly illustrates the boundaries between regular and context-free languages using practical, executable examples.