॥ सा विद्या या विमुक्तये ॥

भारतीय प्रौद्योगिकी संस्थान धारवाड़

Indian Institute of Technology Dharwad

Counter-Free Automata & Nerve Net Simulation

An R&D Project (Autumn 2023) by
B Siddharth Prabhu (200010003) & Cheedrala Jaswanth (200010008)

Project Advisors:
Prof. Ramchandra Phawade,
Prof. Rajshekhar K.

Outline

- Aim of the R&D Project
- Literature Review
- Methodology - Part I
- Methodology - Part II
- Conclusion and Future Scope
- Deliverables/Demo

Aim of the R&D Project (Problem Statement)

In the initial RnD project proposal:

- Construction of a syntactic monoid for a given automaton
- Checking if a monoid is aperiodic
- Checking if a DFA is counter-free
- Nerve Nets Simulator

Effectively:

- Aperiodicity-checker (with syntactic monoid and counters)
- Nerve Nets Simulator

Literature Review

- Papers listed in our report.
- Authors include Kleene, Burks, Wright, etc. — mostly from the 50’s, 60’s and 70’s.
- Review of GAP documentation (old and current).
- *Nerve Nets*: term misused for neural networks at times

Counter-free automata: Methodology

Step 1

GAP Helper Functions

`SyntacticSemigroup`

`IsAperiodicSemigroup`

(Detect if Periodic or Aperiodic)

Step 2

Get Monoid Elements

`RatExpToAut`

`TransitionSemigroup`

`GeneratorsOfSemigroup`

`SemigroupByGenerators`

`MonoidByGenerators`

Step 3

Periodicity-based operations

`MultiplicationTable`

`If Periodic:`

- `BFS-code*`
- `Save Monoid after Disambiguation`

Step 4

(If Periodic,) Counter Identification

`graphchecker.py`*

- Check loops when permuting over a transformation.
- Due to disambiguation of elements, word can be identified
- Mod-N counter

BFS-Style Approach to disambiguation

```
open := [];
closed := [];
hash := HashMap();
for ele in real_gen do
    Append(open, [ele]);
    hash[ele] := [Position(real_gen, ele)];
od;
while (not IsEmpty(open)) do
    curr := Remove(open,1);
    curr_path := hash[curr];
    for ele in real_gen do
      child := curr*ele;
       temp := ShallowCopy(curr_path);
       Append(temp, [Position(real_gen, ele)]);
        if ((not (child in Keys(hash)))
         and (not (child in real_gen))
         and (not (child = curr)))
         then
           hash[child] := temp;
        fi;
        if ((not (child in open))
          and (not (child in closed))
          and (not (child = curr))
          and (not (child in real_gen)))
        then
           Append(open, [child]);;
        fi;
    od;
    Append(closed, [curr]);
od;
```

In brief, we branch until there is repetition.

Graph Checking: A solved example

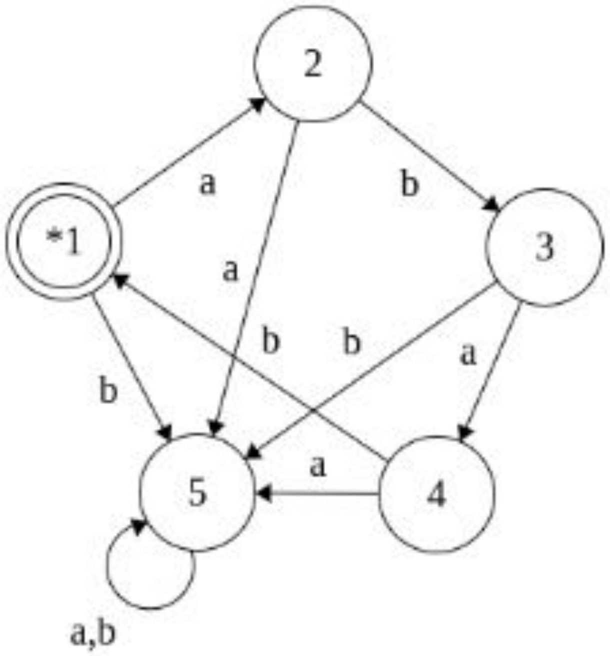


Figure 1: Automaton for $(abab)^*$

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 5 & 5 \end{pmatrix}$$

$$\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 5 & 1 & 5 \end{pmatrix}$$

$$\sigma\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 5 & 5 \end{pmatrix}$$

$$\psi\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 5 & 2 & 5 \end{pmatrix}$$

$$\psi^2 = \sigma^2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$\sigma\psi\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 2 & 5 & 5 \end{pmatrix}$$

$$\psi\sigma\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 5 & 3 & 5 \end{pmatrix}$$

$$\sigma\psi\sigma\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 3 & 5 & 5 \end{pmatrix}$$

$$\psi\sigma\psi\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 5 & 1 & 5 \end{pmatrix}$$

Graph Checking: Permutation Graphs

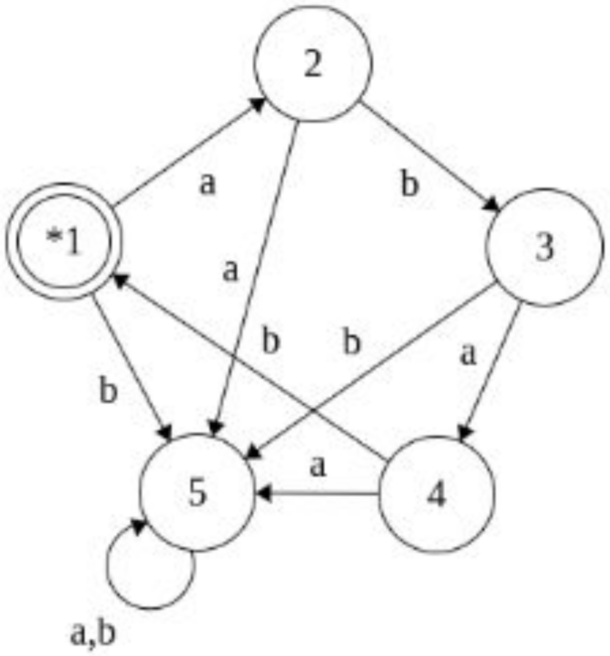
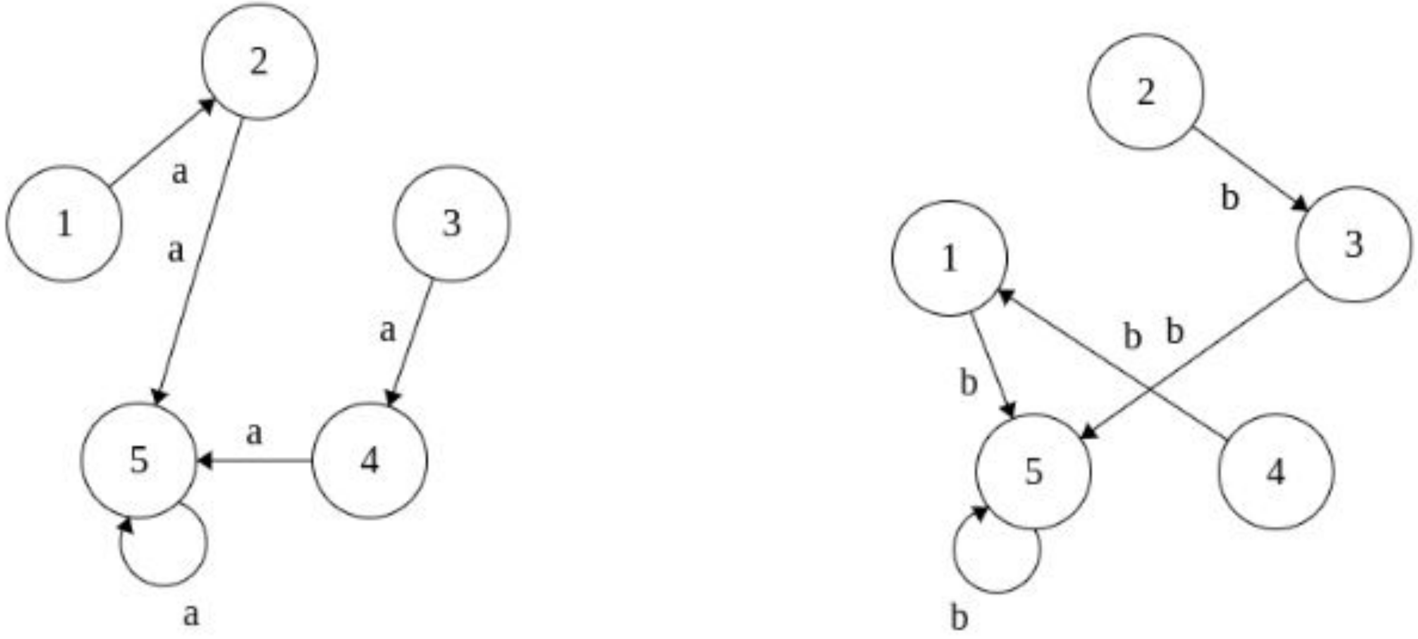


Figure 1: Automaton for $(abab)^*$



(a) Permutation graph for a

(b) Permutation graph for b

Figure 2: Some permutation graphs with no loops detected

Graph Checking: Permutation Graphs

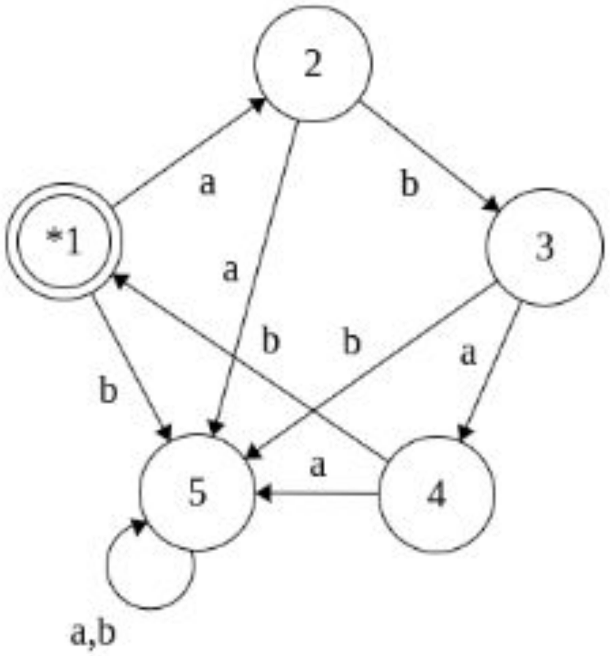
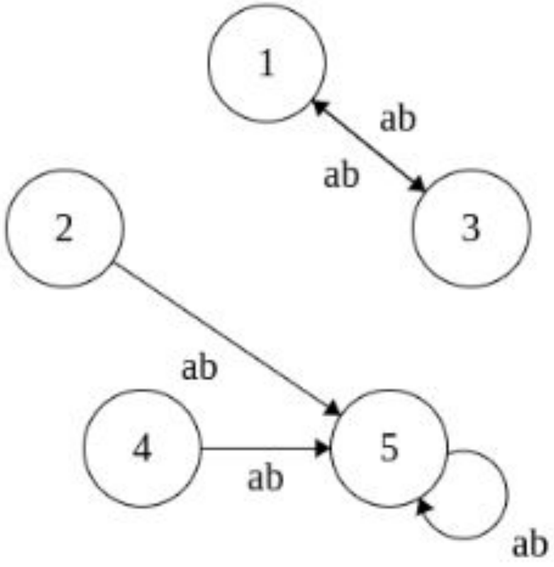
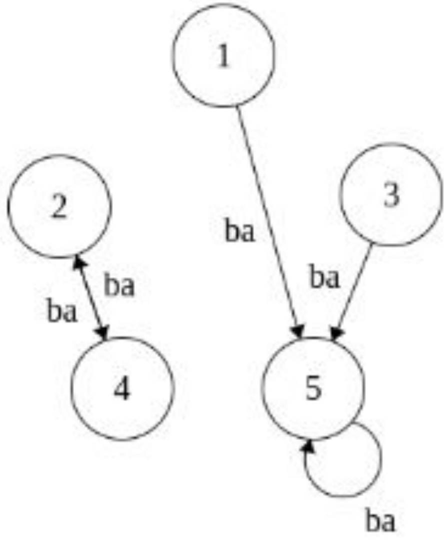


Figure 1: Automaton for $(abab)^*$



(a) Permutation graph for ab



(b) Permutation graph for ba

Figure 3: Some permutation graphs with loops detected

Observations from the CFA tool

- Loops obtained when permuting over elements of the monoid are directly related to periodicity Mod-N.
- List of real-values of regex were verified to be aperiodic. (`shortlist.txt, longlist.txt`)
- (abab)* has mod-2 counters of ab and ba.
- Similarly, (abcabc)* has mod-2 counters of abc, bca, cab.

Demo - 1

Using the aperiodicity checker tool

Nerve Net: Mathematical Specification

To simulate the working of nerve nets, we must formalize its structure mathematically, similar to the $(Q, \Sigma, \delta, q_0, F)$-definition of automata, or the (N, Σ, R, S) of Context-free grammars. We first consider the set of neurons N, and set of axons A. Let the set of user-inputs be I, and the set of outputs (to the user) be O. Each axon a corresponds to a tuple defined as (S_a, E_a, D_a, N_a), where:

- S_a refers to the start-point of axon a, where $S_a \in (N \cup I)$, since the start-point of an axon can be a neuron or a user-input. (Note: $I = \{I1, I2, ...\}$ in our simulation.)
- E_a refers to the end-point of axon a, where $E_a \in (N \cup O)$, since the end-point of an axon can be a neuron or a user-output. (Note: $O = \{O1, O2, ...\}$ in our simulation.)
- D_a is the number of delay elements on axon a, and is a non-negative integer value for all a.
- N_a refers to the nature of the axon upon incidence to a neuron, and can be excitory (1) or inhibitory (0).

[Snippet of the Report]

Nerve Net Simulation: An Overview

- One attempt: from automata via decoded normal form nets.
- Shall not bridge the gap right now, since it requires knowledge of many kinds of (and derivatives of) nerve nets. ($\rightarrow$ future scope)
- For now, we can simulate a nerve net given its specifications, and an input string.
- Link to GitHub Repository in report.

Different Approaches to Simulation

- Dictionary-Based *[Current Approach]*
- Class-Based
- Graph-Based (`networkx`, `graphviz`)

Demo - 2

- Code Overview
- Simulating a Nerve Net

Conclusions and Future Scope

- Scalability: to be considered (monoid disambiguation, OOPS-based nerve net)
- Nerve Net Variants
- Next Avenue: Visual approaches to simulation

[Q/A]

Thank You!