CS309: RnD Report
Counter-Free Automata and Nerve Nets

B Siddharth Prabhu
200010003@iitdh.ac.in

Cheedrala Jaswanth
200010008@iitdh.ac.in

November 24, 2023

1 Introduction

This is an informal report that serves to simply summarise and present all of the ideas and code developed during the course of the Research and Development project (carried out in Autumn 2023) in a contiguous format. Although not mandatory for an RnD project, we hope that this report provides all necessary information for anyone wishing to use the developed tools, or for any further developments in this direction. We assume that the reader is acquainted with the basics of automata and group theory, up to the level of the courses CS202 and CS203 at IIT Dharwad.

2 Terms and Definitions

Below are some important terminologies related to the project:

Monoid: An algebraic structure consisting of a set equipped with an associative binary operation and an identity element. (Note that all groups are monoids.)

Semigroup: An algebraic structure consisting of a set together with an associative internal binary operation on it, i.e., a monoid without identity.

Syntactic Monoid: It is the semigroup associated with the minimal automaton of a language.

Transformation Semigroup: It is a collection of transformations that is closed under function composition. If it includes the identity function, then it is a *transformation monoid*.

Multiplication Table of a Monoid: A visual representation of the binary operation defined on a monoid, in a tabular format.

Subgroup of a Monoid: (As per our observations,) A loop obtained by permuting on the elements of a monoid using its multiplication table. More on this is in Section (5). Informally, we point out this loop in the 'permutation graph' of the monoid.

Periodic Automaton: If there are non-trivial subgroups in the monoid associated with an automaton, then the automaton is said to be periodic.

Mod-N Counter: An automaton is said to contain a Mod-N counter for a string/word w, if it is able to count the number of occurrences of the word w Mod-N, by virtue of the loops contained in its permutation graph. (Example: $(abab)^*$ contains a Mod-2 counter for the word ab.)

Nerve Net: It is a finite set of input axons and neurons, where each neuron has as part of it one or more axons, and each axon is incident to at most one neuron. Each axon is either inhibitory or excitory.

Loop in a Nerve Net: A loop is a sequence $N_0A_0N_1A_1...N_{p-1}A_{p-1}N_p$ where each A_j is an axon of the neuron N_j and incident to the neuron N_{j+1}, and where $N_0 = N_p$ but otherwise $N_i \neq N_j$ for $i \neq j$.

Well-formed Nerve Net: A nerve net in which, for every loop of axons, $\sum_{i=0}^{p-1} A_i \geq 1$. In other words, there is no instantaneous feedback in any loops.

3 General direction of the project

The project grew out of the idea that there is a class of nerve nets that are equivalent to aperiodic automata. In particular, the set of events representable by 'loop-free' nerve nets (LF) are said to be equivalent to the set of non-counting events (NC).

- **Part 1:** To check the aperiodicity of an automaton, and generate other info that describes an automaton. (Such as the monoid corresponding to the automaton's syntactic semigroup, the multiplication table for the monoid, the regex, and the images of the automaton and subautomaton.)
- **Part 2:** To formally define nerve nets (for implementation), and to simulate a nerve net, given the specifications
- **Part 3:** To bridge the gap between nerve nets and automata in such a way that the aperiodicity of an automaton is evident from the loop-free-ness of the corresponding nerve net. (*This part led to a few tangents on different constructions, and is a topic for further research and review.*)

APIs for subautomaton plotting like `VisualDFA`, `graphviz`, and `networkx` were experimented with, but they would require some extensive work in input formatting. Working using GAP's built-in automaton drawing functions is not suggested due to its vague documentation and inconsistent outputs.

4 Aperiodicity and the GAP API

Our GAP-based Aperiodicity checker is available in this GitHub repository, and requires characters to be delimited with angular brackets (to ensure compatibility with multi-symbol characters). This tool effectively 'pipelines' functions available in the GAP API (with a loose definition of 'pipeline'), combining them to dump all relevant automata-information into a user-defined location. Main elements of this process include:

- Taking the regular expression `rat` as input, the `SyntacticSemigroupLang(rat)` function is used to obtain the corresponding syntactic semigroup. Similarly, there are functions to get the automaton associated with the regex. Then, `IsAperiodicSemigroup()` is used to check whether this semigroup is aperiodic. (Note that *transition semigroup* refers to the semigroup of a minimized automaton; not the automaton itself.)
- **If aperiodic,** then the monoid elements are obtained from the semigroup generators, following which a multiplication table is constructed.
- **If periodic,** then we would like to know the *word* responsible for the loop, and the value of N for which the counting of the word takes place. So, after the monoid and corresponding multiplication table are computed, we must *disambiguate* the elements of the monoid.
- To disambiguate the monoid elements, one approach is to get into the internals of GAP and figure out how the monoid is stored; it would make sense for this to happen systematically, and for there to be ways to access these structures. However, upon a week or two of efforts, we found a BFS-esque (Breadth-First Search) disambiguation process to be sufficient for the purposes of the project. On running this BFS, we can deduce the word corresponding to each element of the monoid (or rather, transition semigroup).
- Next, a permutation graph has been constructed (in `graphchecker.py`) to check whether the graph corresponding to permuting over each monoid element contains loops. If such loops are found, then the word can be identified, and the value of N is the size of the loop.

It may be possible for more work to be done in figuring out the data structures storing the monoid and its multiplication table, though this would require lots of patience; the GAP system is largely coded in GAP itself, with numerous `.gi` files, and is thus hard to navigate. A GAP-based monoid disambiguation technique would be greatly helpful, by eliminating the BFS-enumeration.

5 Periodicity: A solved example

Consider the following automaton which represents $(abab)^*$. The initial state is denoted by *.

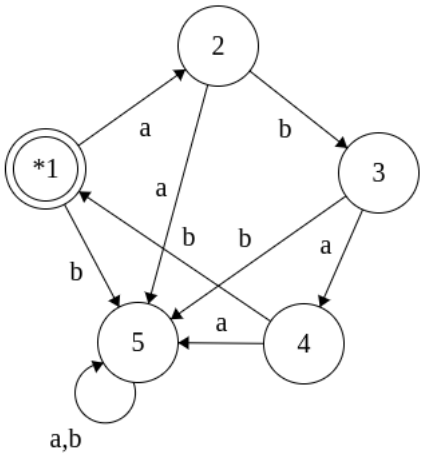


Figure 1: Automaton for $(abab)^*$

Consider $a = \sigma$ and $b = \psi$. Then, we compute the permutations and their graphs as follows:

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

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

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

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

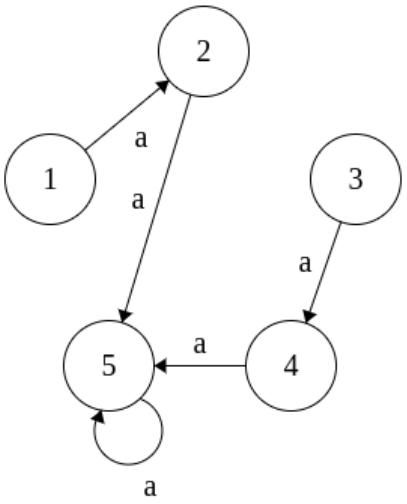
$$\psi^2 = \sigma^2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

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

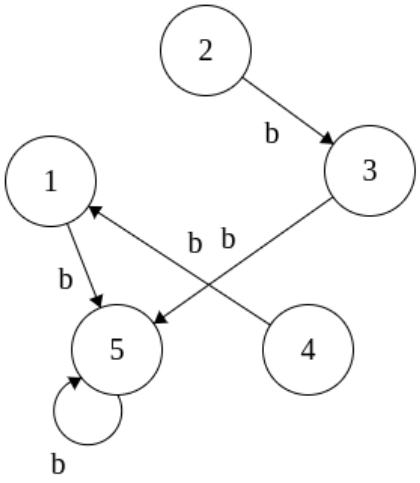
$$\psi\sigma\psi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 5 & 3 & 5 \end{pmatrix}$$

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

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

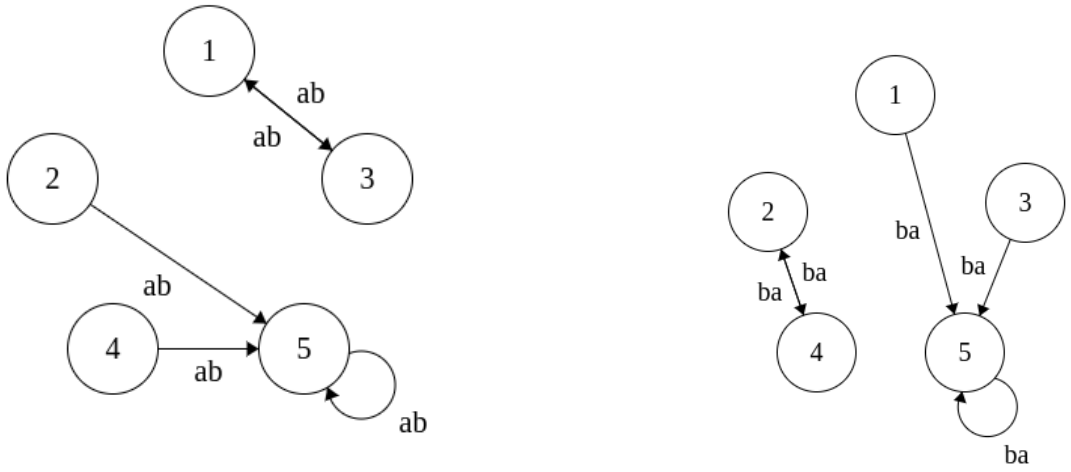


(a) Permutation graph for a



(b) Permutation graph for b

Figure 2: Some permutation graphs with no loops detected



(a) Permutation graph for ab

(b) Permutation graph for ba

Figure 3: Some permutation graphs with loops detected

In the above graphs for ab and for ba, there is a cycle of length 2. This corresponds to a mod-2 counter in the automaton. The words ab and ba are each responsible for Mod-2 counters in the above automaton for $(abab)^*$, as shown in the permutation graphs.

6 Aperiodicity: Observations

Below are observations made using (and ideas developed during the implementation of) our tool:

- Loops obtained when permuting over elements of the monoid are directly related to periodicity Mod-N. This makes sense because each run of a word w in the permutation graph denotes the possible movements from each state on application of the word w. If there exists some n for which all states loop back to themselves on a run of w^n, then there is quite understandably a loop on w Mod-n.
- We were provided a list of regular expressions that corresponded to some real system-based values, linked to 'properties' in runtime verification. These expressions were suggested to be aperiodic, and were verified to be so, using the Aperiodicity checker, with outcomes summarized in a spreadsheet.
- It is well-known that the expression $(abab)^*$ contains a Mod-2 counter for the word ab. However, there is also a counter observed for ba. Curiously enough, the loop formed in the permutation graph of ba does not pass through the start state, which may point to it being a (very loosely defined) semi-counter. It is still countable Mod-2, albeit with some overhang (or underhang). Similarly, $(abcabc)*$ contains Mod-2 counters for abc, bca, and cab respectively.

7 Structure and Working of Nerve Nets

Nerve Nets, as defined earlier, are finite sets of axons and neurons. The axons are either *excitory* (solid dot) or *inhibitory* (hollow dot), and the criterion for 'firing' of a neuron, i.e., generation of a high-signal on axons emanating from a given neuron, is as follows:

$$\#Exc_{high}[N_i] - \#Inh_{high}[N_i] \geq Th[N_i]$$

In other words, for a given neuron N_i, the number of active excitory incident axons $\#Exc_{high}$ must exceed the number of active inhibitory incident axons $\#Inh_{high}$, with the difference being greater than or equal to the neuron's threshold Th, in order for it to fire a high signal onto outbound axons.

We are well-acquainted with the notion of ***acceptance*** of words of a language in automata, but it is a tad bit different for nerve nets. For nerve nets, we consider the notion of junctions, which are points on axons that may or may not be fired at any given time instant. Each junction can be associated with an *event*. If we pick one such junction to be an 'output line,' then the event described by this junction would correspond to the set of words that are considered *accepted*. Consider the following nerve net, presented in Chapter 6 of Patel's thesis, shown below:

Figure 4: Nerve net with event at D being $\Sigma^*(1 \cup 3)\Sigma^*2$

While not extensively explained in the paper itself, the input event encoding is as follows: $0 : (C : 0, B : 0);\ \ 1 : (C : 0, B : 1);\ \ 2 : (C : 1, B : 0);\ \ 3 : (C : 1, B : 1)$, and $\Sigma = \{0, 1, 2, 3\}$. Hence, D is high if the current input is 2, and the input lines have been 1 or 3 at some point in the past, in the input so far. Neurons have no inherent delay in this construction. Delays are enforced via delay elements denoted using vertical lines perpendicular to the axons. For an axon with d delays, it can be perceived as holding $d + 1$ values due to the $d + 1$ junctions present on the axon, each with a defined 'event.' Nerve nets must be **well-formed** so as to avoid instantaneous feedback, with every loop containing at least one delay element.

For the above example, if the input stream is 0132, then the following table displays the values of B, C, and D respectively.

Time t	Input	B_t	C_t	D_t
1	0	0	0	0
2	1	1	0	0
3	3	1	1	0
4	2	0	1	1 (Success!)

Table 1: Junction values over time for input 0132

8 Mathematical Representation and Encoding of Nerve Nets

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).

9 Useful Nerve Net Widgets

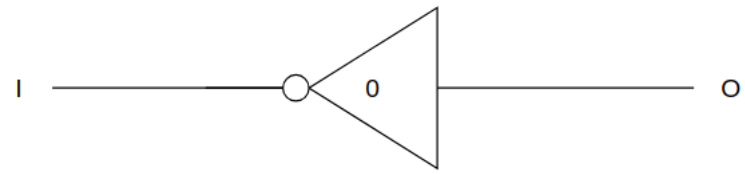


Figure 5: NOT gate ($O = \overline{I}$)

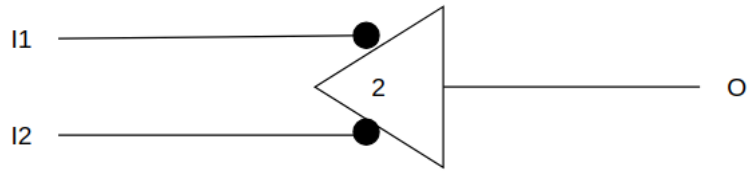


Figure 6: AND gate ($O = I1 \wedge I2$)

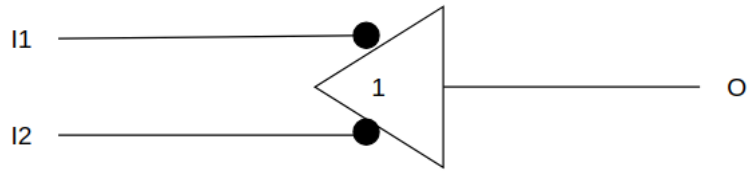


Figure 7: OR gate ($O = I1 \vee I2$)

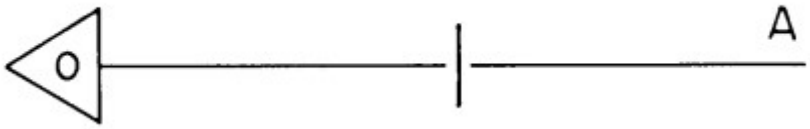


Figure 8: Initial Pusher (high at $t = 0$ and low otherwise)

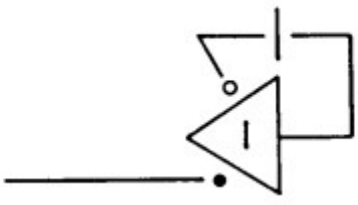


Figure 9: Buzzer (intermittent pulses)

Above are some useful widgets for use in nerve nets, including basic logic gates, a buzzer, and initial push. The logic gates allow us to permit a union or intersection (or disallow) specific inputs. A use case of such widgets is presented in the following section.

10 Decoded Normal Form Nets: A Brief Account

Burks and Wang, in their publication *The Logic of Automata* explore different constructions of nerve nets from automata, of which one is the **decoded normal form net**. This did not turn out to be useful in detecting aperiodicity, due to its great similarities with the very source automaton it is generated from, but is nevertheless an interesting topic. Thus, we shall take a glance at this visual approach of automaton-to-nerve net translation.

We generally construct transition tables for automata with states on the vertical and letters on the horizontal, with each entry denoting the final state upon application of a letter from a given start state. But what if we flipped it to have states on both axes of the table, with inputs in between? We would then have a square matrix ($n \times n$) as follows:

$$\delta_{n=4} = \begin{pmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{pmatrix}$$

Here, I_{ij} represents the input (or set of inputs) necessary to move from state i to state j. It is possible that for some (i, j), $I_{ij} = \emptyset$, if there is no edge joining them. We have seen in previous sections (indirectly), that it is possible to encode inputs and sets of inputs (using AND). Hence, a subnet could be created for each of the I_{ij}'s, and then could be laid out in a matrix-box format, as shown in the following image from Burks and Wang's paper.

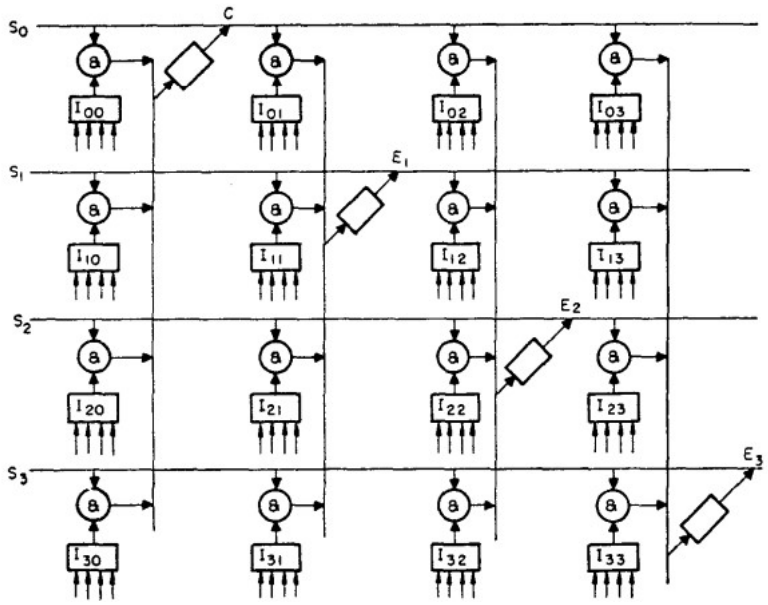


Figure 10: Matrix box with $n = 4$

We must give an 'initial push' to the line corresponding to the start state, and must observe the values of each of the lines. By design, only one state line (horizontal) would be high (0) at any given time instant, while the rest would be low (1).

Clearly, this construction preserves the structure of the automaton, and thus replicates any loops present in the automaton. However, not all loops in automata indicate counters. Thus, we may obtain loops in decoded normal form nets for automata that are aperiodic, such as $(ab)^*$. Hence, this construction technique is inadequate for our goal of mapping aperiodic automata to loop-free nerve nets.

11 Nerve Net Simulator Guide

The code for nerve net simulation is up and available at this GitHub repository. It is currently not integrated with automata, and behaves as a standalone machine. Inputs must be given as streams

of equal lengths in `config.json`, and must be given keys as `I1, I2, I3,...`. Input files must be of the following format:

- Line 1: Number of input lines (I1, I2, ...)
- Line 2: Number of output lines (O1, O2, ...)
- Line 3: Number of neurons (1, 2, ...)
- Line 4: Threshold values for neurons ordered by neuron ID
- Line 5 onwards: Axon specifications, in the following format: `<start-point> <end-point> <delay> <nature>` , where:
  - `<start-point>` can be a neuron or an input line.
  - `<end-point>` can be a neuron or an output line.
  - `<delay>` is a non-negative integer.
  - `<nature>` is either 0 (inhibitory) or 1 (excitory).

Translating meaningful inputs into bits is left to the next-layer developer, who can perhaps convert their queries into bitstreams as required for this program. To run the program, use `python3 src/simulator.py` .

12 Conclusions and Future Scope

Future scope and work to be done in this area include:

- Examining the internals of GAP data structures could help make our implementations more scalable, by perhaps eliminating the enumerative breath-first search-based approach.
- There are numerous types, names, and definitions (and derivatives) of nerve nets, each with different constructions. Only some of them may be useful in our search for evident aperiodicity.
- An algorithm to convert aperiodic automata into loop-free nerve nets would connect the two parts of our project, and could help expedite runtime verification.
- If there is a reliable algorithm to 'reduce' nerve nets or transform them into equivalent forms, then perhaps, the loops in decoded normal form nets could be eliminated for aperiodic expressions.
- We made attempts at simulating nerve nets with classes and objects, and also with a graphical approach (using `graphviz`/`networkx`). These were not fully fruitful, but if fully developed, could lead to more OOPS-oriented, visualizable nerve net implementations.

In terms of conclusions made, most have been included in previous sections. But, here are other takeaways that we have had.

- The documentation of softwares (like GAP) may not always be easily parsable (for a person), and it is sometimes better to deduce things from outside the code, rather than deepdive through obsolete libraries (that have been incorporated into recent ones, but the point still stands).
- Parsers like *Lark* and *Yacc/Bison* make text-parsing and user-input more modular and clean.
- There are still many concepts out there that have not been implemented in code; ideas developed even 50-60 years ago (like nerve-nets) do not have libraries or common APIs. (Though, there are APIs like GAP with relatively niche but useful functions.)
- Working out an example by hand, and defining parameters for a piece of code manually is essential to smoothly implementing the code. This would be hard to do if the set of usable functions is small, but is still a good approach to ensure structured code.

13 Miscellanea

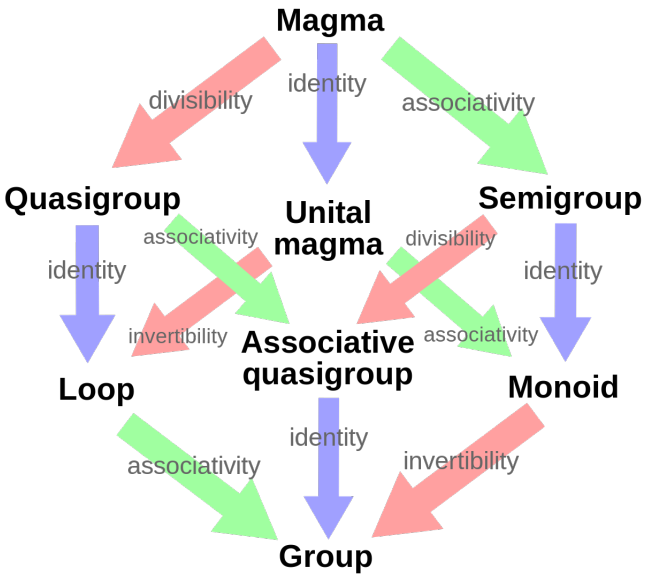


Figure 11: Algebraic Structures

References

[1] N. ALON, A. K. DEWDNEY, AND T. J. OTT, *Efficient simulation of finite automata by neural nets*, J. ACM, 38 (1991), p. 495–514.

[2] A. W. BURKS AND H. WANG, *The logic of automata—part i*, J. ACM, 4 (1957), p. 193–218.

[3] A. W. BURKS AND H. WANG, *The logic of automata—part ii*, J. ACM, 4 (1957), p. 279–297.

[4] A. W. BURKS AND J. B. WRIGHT, *Theory of logical nets*, Proceedings of the IRE, 41 (1953), pp. 1357–1365.

[5] A. W. BURKS AND J. B. WRIGHT, *Theory of logical nets*, Journal of Symbolic Logic, 19 (1954), pp. 141–142.

[6] P. CULL, *Neural Nets: Classical Results and Current Problems*, vol. 1, Oregon State University, Corvallis, OR, Oregon State University, Dept. of Computer Science, 1995.

[7] S. KLEENE, *Representation of events in nerve nets and finite automata*[1], Automata Studies: Annals of Mathematics Studies. Number 34, (1956), p. 3.

[8] W. S. MCCULLOCH AND W. PITTS, *A logical calculus of the ideas immanent in nervous activity*, The Bulletin of Mathematical Biophysics, 5 (1943), pp. 115–133.

[9] R. MCNAUGHTON AND S. A. PAPERT, *Counter-Free Automata (M.I.T. Research Monograph No. 65)*, The MIT Press, 1971.

[10] T. MITOMA AND K. SHOJI, *Syntactic monoids and languages (algebra, languages and computation)*, Institute of Mathematical Analysis Kokyuroku, 1437 (2005).