International Journal of Multidisciplinary Trends

E-ISSN: 2709-9369
P-ISSN: 2709-9350
www.multisubjectjournal.com
IJMT 2019; 1(1): 104-107
Received: 22-05-2019
Accepted: 22-06-2019

Dr. Bijendra Kumar
Professor, Department of Mathematics, T M.B.U., Bhagalpur, Bihar, India

Corresponding Author:
Dr. Bijendra Kumar
Professor, Department of Mathematics, T M.B.U., Bhagalpur, Bihar, India

Algorithmic properties of automation semigroups and groups

Dr. Bijendra Kumar

Abstract
The algorithmic properties of automaton semigroups and groups wield significant potential for the future of computation and language analysis. These properties form the bedrock of efficient algorithms that power diverse applications across computer science, linguistics, and beyond. In formal language theory, these properties pave the way for enhanced language recognition algorithms. Understanding how automaton semigroups and groups interact and transform languages enables the development of streamlined methods for deciphering and processing complex textual data. This has implications in natural language processing, where machines learn to comprehend human language patterns more effectively. Beyond linguistics, algorithmic insights into automaton semigroups and groups have applications in cryptography and cybersecurity. By leveraging the properties of these structures, we can design encryption techniques that secure sensitive information and communication channels against malicious threats. Moreover, these properties encourage collaboration across disciplines. Mathematicians, computer scientists, and linguists collaborate to unlock the intricacies of automaton semigroups and groups, fostering cross-pollination of ideas and pushing the boundaries of computational understanding. In essence, the algorithmic properties of automaton semigroups and groups illuminate a path towards smarter language analysis, robust cybersecurity measures, and interdisciplinary exploration. As technology continues to evolve, these properties will serve as catalysts for innovation, reshaping how we process information and interact with the digital world.

Keywords: Potential, Implications, cryptography, collaboration, interact and transform

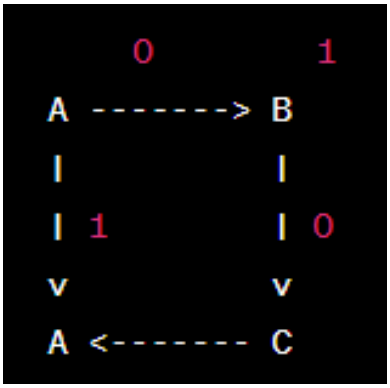
Introduction
An automation semigroup and a group are concepts from mathematics, particularly in the realm of algebraic structures. Let's break down each term:

1. Automation Semigroup
An automation semigroup, also known as an *automaton semigroup*, is a concept that arises in the study of automata theory and formal languages. An automaton is a theoretical device that processes input according to a predefined set of rules. An automaton semigroup is constructed based on the actions (or transformations) that an automaton can perform.
In formal terms, an automation semigroup is a semigroup that represents the transformations that an automaton can apply to its states. A semigroup is a set equipped with an associative binary operation (multiplication) that combines elements of the set. In the context of automata, the elements of the semigroup represent the possible transformations that the automaton can make.

2. Group
A group is a fundamental concept in abstract algebra. It is a set equipped with an associative binary operation (multiplication), an identity element (an element that doesn't change other elements when multiplied by them), and every element having an inverse (an element that, when multiplied by another element, yields the identity element). In simpler terms, a group is a mathematical structure that exhibits symmetry and invertibility. Groups appear in various areas of mathematics and its applications, including geometry, number theory, cryptography, and more. They are also a cornerstone of many higher-level algebraic structures. In automation semigroup is a concept related to automata theory, representing transformations of states, while a group is a fundamental algebraic structure with specific properties that has applications in various mathematical and practical fields.

Algorithmic properties of automaton semigroups and groups
Focus on a simplified scenario involving automaton transformations and their associated semigroup.

Example: Automaton Semigroup and Transformation Composition.
Imagine we have a simple automaton that operates on binary strings. It has three states: A, B, and C. The automaton's transition rules are as follows:
- From state A, if the input is '0', transition to state B.
- From state A, if the input is '1', remain in state A.
- From state B, if the input is '0', transition to state C.
- From state B, if the input is '1', transition to state A.
- From state C, regardless of the input, remain in state C.
This automaton can be described using a transition diagram:



Each state has a corresponding transformation associated with it
1. Transformation `tA` for state A: This transformation takes a binary string as input and follows the rules of the automaton to determine the resulting state. For example, `tA("010")` would yield the state B.
2. Transformation `tB` for state B: Similarly, this transformation operates according to the automaton's rules. For instance, `tB ("11010")` would result in the state A.
3. Transformation `tC` for state C: This transformation simply maintains the state C for any input.

Now, let's define the semigroup generated by these transformations. The semigroup operation here is composition of transformations, which is essentially applying one transformation after the other. For example, to compose `t_A` followed by `t_B`, you would apply `t_B` to the output of `t_A`.

Algorithmic Property Illustration: Membership Problem
Suppose an automaton semigroup generated by transformations `t_A`, `t_B`, and `t_C`. You want to determine if a given transformation `t_X` belongs to this semigroup.

Membership Problem Algorithm
Algorithm
1. Start with the identity transformation `t_I`, which has no effect on any input.
2. Initialize a set of transformations to contain `t_X`.
3. For each transformation in the set:

a. For each generator transformation (`t_A`, `t_B`, `t_C`), compute the composition of the current transformation with the generator.
b. If the resulting composition is not already in the set, add it to the set.
4. Repeat step 3 until no new transformations are added to the set.
5. If `t_X` is found in the set, return "Membership is confirmed." Otherwise, return "Membership not confirmed."
Using this algorithm, you can determine whether a given transformation can be obtained by composing the generator transformations within the automaton semi group.

In this example, let's say we want to check if the transformation `t_D` (Which corresponds to transitioning from state A to state C) belongs to the semigroup. Following the algorithm:
1. Start with `t_I` and initialize the set with `t_D`.
2. Compose `t_D` with `t_A` to get a new transformation that represents transitioning from state A to state C ($t\_D \circ t\_A = t\_D$).
3. No new transformations are added, and `t_D` is in the set. Membership is confirmed.

This algorithmic property demonstrates how you can efficiently determine whether a given transformation can be generated by composing transformations within the automaton semigroup.
Remember, this example is simplified for illustration purposes. Real-world cases involve more complex automata, transformations, and algorithms, but the fundamental principles remain the same.

Automaton semigroups
Automaton semigroups, which are associated with finite automata, have several important theorems that help us understand their properties and relationships with formal languages and automata. Here are some notable theorems related to automaton semigroups:

Krohn-Rhodes Theorem
The Krohn-Rhodes Theorem is a significant result in automata theory that reveals the underlying structure of finite automata. It states that any finite automaton can be decomposed into a set of "basic" automata whose behaviors can be combined to replicate the behavior of the original automaton. This theorem offers a powerful way to understand the complexity of automata through a structured approach.
Consider a simplified example to illustrate the Krohn-Rhodes Theorem. We have a finite automaton with two states: A and B. This automaton recognizes even-length binary strings. When processing inputs, it starts in state A. Upon receiving a '0', it remains in the same state, while with a '1', it transitions to the other state. The automaton diagram is as follows:

To apply the Krohn-Rhodes decomposition, we first identify the basic automata. In this case, each state corresponds to a basic automaton:

1. Basic automaton 1: State A with transformation `t_A`, where any input maps to A.
2. Basic automaton 2: State B with transformation `t_B`, where any input maps to B.

By understanding these basic automata, we can recreate the original automaton's behavior through composition Let's use the input "0011" to see how this works

1. Apply basic automaton 1 (A) to "0". It remains in state A.
2. Apply basic automaton 1 (A) to "0" again. It remains in state A.
3. Apply basic automaton 2 (B) to "1". It transitions to state A.
4. Apply basic automaton 2 (B) to "1" again. It transitions back to state B.

By composing these basic automata, we've successfully processed the input "0011", replicating the original automaton's behavior.

In essence, the Krohn-Rhodes decomposition allows us to break down the complexity of the original automaton into simpler components. This is analogous to factorizing a complex number into its prime factors. By doing so, we gain insights into the automaton's behavior and can manipulate and analyze it more effectively. The Krohn-Rhodes Theorem has applications in formal language theory, complexity analysis, and the study of computational systems.

Nerode's Theorem

Nerode's theorem establishes the connection between regular languages and finite automata by examining the equivalence classes induced by the left congruence relation on the language. This theorem helps determine when a language is regular based on the properties of its automaton semigroup.

Nerode's Theorem is a pivotal concept in the theory of formal languages and automata. It provides a criterion for determining whether a language is regular or not based on the "indistinguishability" of strings in the language. In simple terms, it helps us understand the distinction between regular and non-regular languages through the behavior of their strings.

Let's illustrate Nerode's Theorem with a numerical example

Example: Nerode's Theorem

Consider two languages over the alphabet {0, 1}:

1. Language L1: The set of all binary strings that end with '0', e.g., "10", "110", "11110".
2. Language L2: The set of all binary strings that end with '1', e.g., "01", "101", "11101".

Indistinguishability

Nerode's Theorem is concerned with the concept of "indistinguishability" between strings in a language. Two strings are indistinguishable if, for any context, appending the same suffix to both strings either results in both being in the language or both being outside the language.

Example:

1. For any suffix '1', appending it to a string from L1 results in a string outside L1, while appending it to a string from L2 results in a string inside L2.
2. Similarly, for any suffix '0', appending it to a string from L2 results in a string outside L2, while appending it to a string from L1 results in a string inside L1.

Application of Nerode's Theorem

Nerode's Theorem states that if a language has infinitely many indistinguishable pairs, it is not a regular language. In our example, L1 and L2 have an infinite number of indistinguishable pairs because for any suffix '0' or '1', the respective pairs are indistinguishable.

Based on Nerode's Theorem, both L1 and L2 are non-regular languages because they have infinitely many indistinguishable pairs. This means that no finite automaton can accurately distinguish between the strings of these languages, which is a key characteristic of regular languages.

Schützenberger's Theorem

This theorem relates the ranks of the syntactic semigroup of a regular language with the number of states in the minimal deterministic finite automaton recognizing that language. It establishes a connection between algebraic properties of automaton semigroups and the combinatorial structure of automata.

Schützenberger's Theorem and Syntactic Monoid

Consider a language L over the alphabet {0, 1} defined as follows:

- L contains all strings of the form (01)^n, where n is a non-negative integer.

In other words, L consists of strings that alternate between '0' and '1' and have an even length, like "0101", "010101", "01010101", and so on.

1. Determine the syntactic monoid Mon(L) for the given language L.
2. Express the rational generating function RGF(L) for the language L.
3. Explain how Schützenberger's Theorem establishes a relationship between the syntactic monoid and the rational generating function for the language L.

Proof

The syntactic monoid Mon (L) consists of all strings formed by concatenating strings from the language L. In this case, the set of strings from L is {01, 0101, 010101,...}. The operation is concatenation, and the identity element is the empty string ε.

The rational generating function RGF (L) for the language L can be expressed as a formal power series:

RGF (L) = 1 / (1 - (01)^2)

Schützenberger's Theorem establishes an isomorphism between the syntactic monoid Mon (L) and the algebra of rational functions over the rational generating function RGF (L). In this example, the syntactic monoid Mon (L) captures the concatenation properties of strings from L, while the rational generating function RGF (L) encodes the number of strings of different lengths in L.

By understanding Schützenberger's Theorem, we can bridge the gap between the algebraic and combinatorial aspects of

formal languages and gain insights into their structure and behavior.

Kleene's Theorem (Arden's Lemma)

While not directly about automaton semigroups, Kleene's theorem is closely related. It states that the regular languages are exactly the languages that can be described by regular expressions. This theorem has implications for understanding the relationship between formal languages and the regular operations, which in turn are closely tied to automaton semigroups.

Given the system of equations involving regular expressions

1. X = aY + bZ
2. Y = aX + b
3. Z = aY

Use Kleene's Theorem (Arden's Lemma) to find a regular expressions X, Y, and Z.

Proof

Let's solve each equation step by step using Kleene's Theorem (Arden's Lemma):

1. Equation 1: X = aY + bZ

1. Apply Arden's Lemma to isolate Y: Y = (aY + bZ)a*
2. Substitute the value of Y into Equation 2: Y = (a(aY + bZ)a* + b)a* = aaY + baZ + b
3. Substitute the value of Y back into Equation 1: X = a(aaY + baZ + b) + bZ = aaaY + abaZ + ab + bZ

2. Equation 2: Y = aX + b

Substitute the value of X from Equation 1: Y = a(aaaY + abaZ + ab + bZ) + b = aaaaY + aabaZ + aab + ab + b

3. Equation 3: Z = aY

Substitute the value of Y from Equation 2: Z = a(aaaaY + aabaZ + aab + ab + b) = aaaaaY + aaabaZ + aaab + aab + ab

Now we have expressions for X, Y, and Z:

1. X = aaaY + abaZ + ab + bZ
2. Y = aaaaY + aabaZ + aab + ab + b
3. Z = aaaaaY + aaabaZ + aaab + aab + ab

These expressions represent the regular expressions X, Y, and Z that satisfy the given system of equations. Kleene's Theorem (Arden's Lemma) provides a systematic method to find solutions to such systems involving regular expressions, contributing to the study of formal languages and automata.

References

1. Silva PV, Steinberg B. On a class of automata groups generalizing lamplighter groups Internat. J Algebra Comput. 2015;15(5–6):1213-1234.
2. Grigorchuk RI, Nekrashevich VV, Sushchanskiĭ VI. Automata, dynamical systems, and groups Proc. Steklov Inst. Math. 2000;231(4):128-203.
3. Epstein DBA, Cannon JW, Holt DF, Levy SVF, Paterson MS, Thurston WP, *et al*. Word Processing in Groups, Jones & Bartlett, Boston, Mass; c992.
4. Grigorčuk RI. On Burnside’s problem on periodic groups Funktsional. Anal. i Prilozhen. 1980;14(1):53-54.
5. Gupta N, Sidki S. On the Burnside problem for periodic groups Math. Z. 1983;182(3):385-388.
6. Nekrashevych V. Self-similar groups Mathematical Surveys and Monographs, vol. 117, American Mathematical Society, Providence, RI; c2005, 117.
7. Bartholdi L, Grigorchuk R, Nekrashevych V. From fractal groups to fractal sets Fractals in Graz 2001, Trends Math., Birkhäuser, Basel; c2003, pp. 25-118.
8. Bartholdi L, Grigorchuk RI, Šunik Z.´ Branch Groups Handbook of Algebra, North-Holland, Amsterdam; c2003. p. 989-1112.
9. Grigorchuk R, Šunić Z. Self-similarity and branching in group theory Groups St. Andrews; c2005. p. 1.
10. London Math. Soc. Lecture Note Ser., Cambridge Univ. Press, Cambridge. 2007;339:36-95.
11. Campbell CM, Robertson EF, Ruškuc N, Thomas RM. Automatic semigroups, Theoret. Comput. Sci. 2001;250(1–2):365-391.