Increasing the Computational Power of n-th Order Limit Languages up to Context-Sensitive Language through Grammar Modifications

Siti Hajar Mohd Khairuddin
Centre for Mathematical Sciences, University Malaysia Pahang Al-Sultan Abdullah

Muhammad Azrin Ahmad
Centre for Mathematical Sciences, University Malaysia Pahang Al-Sultan Abdullah

Kavikumar Jacob
Faculty of Applied Science and Technology, University Tun Hussein Onn

Mohd Sham Mohamad
Centre for Mathematical Sciences, University Malaysia Pahang Al-Sultan Abdullah

https://doi.org/10.5109/7395670

出版情報：Proceedings of International Exchange and Innovation Conference on Engineering & Sciences (IEICES). 11, pp.1246-1251, 2025-10-30. International Exchange and Innovation Conference on Engineering & Sciences
バージョン：

Increasing the Computational Power of n-th Order Limit Languages up to Context-Sensitive Language through Grammar Modifications

Siti Hajar Mohd Khairuddin[1], Muhammad Azrin Ahmad[1], Kavikumar Jacob[2], Mohd Sham Mohamad[1],
[1]Centre for Mathematical Sciences, University Malaysia Pahang Al-Sultan Abdullah, Lebuh Persiaran Tun Khalil Yaakob, 26300 Gambang, Pahang, Malaysia, [2]Faculty of Applied Science and Technology, University Tun Hussein Onn, Malaysia
Corresponding author email: azrin@umpsa.edu.my

Abstract: *In previous research, the definition of the n-th order limit language, as introduced by Goode and Pixton, has been refined from a rule's perspective. Initially, the n-th order limit language, denoted as Ln, involved the deletion of transient words in L_(n-1). Subsequent research extended this by emphasizing the quantity of initial strings and rules within the splicing system. However, these findings were derived from a splicing system model aimed at preserving biological splicing characteristics through the process of cutting and pasting genetic material (DNA) in the presence of restriction enzymes and ligase. This study shifts focus to a model based on the Chomsky hierarchy for language generation due to limitations in the biological-based model, which restricts language production to regular language only. Modifications were applied, such as varying the splicing system's variables, to adapt the definition to a widely used framework in language generation research, achieving a higher hierarchy of language production by the splicing system. Previously, the n-th order limit language was generalized using the biological-based splicing system, the Head splicing system S=(A,I,B,C) which was limited to the generation of languages to regular and context-free language only. To address this limitation, a revised definition of the n-th order limit language was formulated using the extended H splicing system γ=(V,T,Am,Rp), characterized by an infinite number of rules and axioms to extend the generation of language to context-free and context-sensitive by modifying grammar. This paper investigates how the new definition, based on the Păun splicing system, enhances the computational power of n-th order limit languages. By imposing a few restrictions and considering various cases, we aim to increase computational power. We have proved that the new definition increases the computational power of the languages produced by the extended H splicing system.*

Keywords: Formal language theory; Splicing system; n-th order limit language; computational power.

1. INTRODUCTION

Formal language theory provides a foundational framework for the study of symbolic sequences, offering key insights into computational models, automata theory, and structural properties of languages [1]. One significant extension of this framework is the splicing system, introduced by Head (1987) in [2], which models the process of DNA recombination through formal string operations. Splicing systems manipulate strings by cutting and recombining based on predefined splicing rules, reflecting natural biological mechanisms such as restriction enzyme activity, ligation, and site-specific recombination [3]. As a result, splicing systems serve as a bridge between theoretical computer science and molecular biology, offering insights into both biocomputing and formal language classification.

Over the past decades, significant research has been dedicated to understanding the computational power of splicing systems. Various models, including finite, circular, iterated, and reflexive splicing systems, have been introduced to explore their expressive capabilities and generative limits. These systems have been used to establish relations with the Chomsky hierarchy, where certain variants have been proven to generate regular, context-free, and even recursively enumerable languages [4]. Notably, some splicing systems have been shown to exhibit Turing universality, suggesting their computational equivalence to classical models such as Turing machines and Lindenmayer systems [5]. Furthermore, the introduction of control mechanisms, weight-based splicing, and probabilistic approaches has expanded the computational scope of splicing systems, making them relevant for applications beyond theoretical studies.

There are some methods to increase the computational power of splicing system based on past studies. Previously, many researchers in this field have increased the computational power of the Turing machine, which the language by introducing a new splicing system inspired from the existing splicing system such as fuzzy splicing system [6], [7], [8], bonded parallel insertion-deletion systems [9], time-varying distributed H (TVDH) system and enhanced time-varying distributed H (eTVDH) system [10], sticker system [11], [12], [13], and probabilistic splicing system [13], [14], [15]. However, some researchers did a number of limitations on the application of rules in the existing splicing system, which would raise the hierarchy of the language produced and overall increase the computational power. For example, by and deletion rules, the higher hierarchy of language will be produced in [9]. Usually, by introducing a new type of splicing system, a few restrictions on the process of splicing to produce language will result in a higher computation power of the language. However, when distinct restrictions are applied to all computing devices and their variations, there are certain limitations on computational power [15].

This study aims to provide a comprehensive analysis of the computational power of splicing systems within the framework of formal language theory. Specifically, the generative capacity in the selected splicing language is examined , placement within complexity classes, and

potential applications in molecular computation and artificial intelligence. By rigorously classifying the computational properties of splicing systems, this work contributes to a deeper understanding of their role as both a biologically inspired computing model and a formal system with broad theoretical and practical significance.

2. LITERATURE REVIEW

Noam Chomsky introduced the Chomsky hierarchy in 1956 as a formal classification of languages based on their generative power and computational complexity [16]. This hierarchy has since become fundamental in theoretical computer science, computational linguistics, and compiler design. The hierarchy consists of four classes: regular languages, context-free languages (CFLs), context-sensitive languages (CSLs), and unrestricted languages, each associated with a specific type of grammar and automaton. The classification of formal language classes is presented below.

2.1 Classification of Formal Languages

Regular languages are the simplest class in the formal language hierarchy. They are defined by regular expressions or generated by regular grammars, where every production rule takes the form $A \to aB$ or $A \to a$, with A and B being non-terminal symbols, and a being a terminal symbol. These languages are recognized by finite automata, which can be either deterministic or non-deterministic. Regular languages exhibit several important properties such as they are closed under operations such as union, intersection, concatenation, and the Kleene star. However, they lack the capability to count arbitrarily long sequences, meaning they cannot recognize patterns like $\{a^n b^n \mid n \geq 0\}$. Common examples of regular languages include strings composed only of the characters 'a' and 'b' that end with the substring "ab", and binary strings that contain an even number of zeros. Regular languages have practical applications in areas such as lexical analysis in compilers and pattern matching in text processing.

Secondly, Context-Free Languages (CFLs) are defined by Context-Free Grammars (CFGs), where each production rule has the form $A \to \gamma$ with A representing a single non-terminal symbol and γ being a string composed of terminals and/or non-terminals. These languages are recognized by pushdown automata, which are computational models that use a stack to provide additional memory. CFLs have several notable properties such as they are closed under union, concatenation, and the Kleene star, but not under intersection or complement. They are particularly well-suited for representing hierarchical structures, such as nested parentheses or HTML tags. Examples of CFLs include the language $\{a^n b^n \mid n \geq 0\}$, which contains equal numbers of 'a's followed by 'b's, and the set of strings consisting of well-formed sequences of parentheses. These languages are widely used in parsing within compilers and in syntax analysis for programming languages.

Thirdly, Context-Sensitive Languages (CSLs) are defined by Context-Sensitive Grammars (CSGs), where the production rules must satisfy the condition $|\alpha| \leq |\beta|$, meaning that the length of the left-hand side (α) is less than or equal to the length of the right-hand side (β). These languages are recognized by Linear Bounded Automata (LBAs), which are Turing machines with memory constrained by the size of the input. CSLs are closed under all standard operations, including union, intersection, complement, concatenation, and Kleene star. They are more expressive than Context-Free Languages but are generally less practical due to their higher computational complexity. A typical example of a context-sensitive language is $\{a^n b^n c^n \mid n \geq 0\}$, which requires equal numbers of the symbols 'a', 'b', and 'c'. Other examples include languages that require contextual validation, such as certain syntax rules found in natural languages. Applications of CSLs include advanced compiler optimizations and complex pattern recognition tasks.

Unrestricted languages, also known as Recursively Enumerable Languages (RELs), are the most powerful class in the Chomsky hierarchy. They are produced by Unrestricted Grammars, which impose no specific constraints on production rules. These languages are recognized by Turing Machines which computational models capable of simulating any algorithm. RELs encompass all computable languages and are the most expressive in the hierarchy. However, they also include undecidable problems, such as the Halting Problem, where no general solution exists to determine if an input will be accepted. Examples include any algorithmically describable language, even if it lacks an efficient recognition method. RELs are essential in theoretical computer science, particularly in studying computation, undecidability and complexity theory. The levels of the Chomsky hierarchy are illustrated in Figure 1.

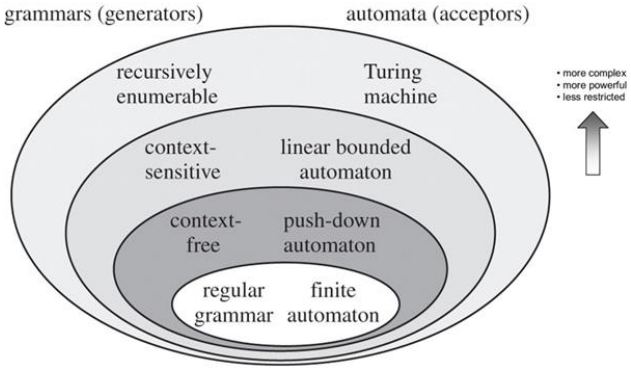

Fig. 1. Chomsky Hierarchy Levels. Source: [17]

Then, grammar plays a fundamental role in formal language theory, serving as a set of rules that define the structure of valid strings within a language [18]. In computational theory, grammars are essential for classifying languages, analyzing their syntactic properties, and designing efficient parsing algorithms. The study of grammars has applications in compiler design, natural language processing (NLP), artificial intelligence, and formal verification.

2.2 Grammar

The concept of formal grammar was first introduced by Noam Chomsky in 1956, leading to the development of the Chomsky hierarchy which defines a framework that classifies grammars based on their generative power and computational complexity. Given below is the hierarchy consists of four main types of grammars.

A. A **Regular Grammar** consists of the following components:

$$G = (N, \Sigma, P, S)$$

Where:

- N = a finite set of non-terminal symbols
- Σ = a finite set of terminal symbols, where $N \cap \Sigma = \emptyset$
- P = A finite set of production rules, following a specific regular form
- $S \in N$ is the start symbol

It consists of a set of production rules where each rule has a single non-terminal symbol on the left-hand side and, on the right-hand side, either a terminal symbol followed by a non-terminal, a single terminal, or the empty string. The rules must conform to a specific pattern, typically either right-linear or left-linear, meaning non-terminals appear consistently on one side of the terminal in each production.

B. A **Context-Free Grammar** is a 4-tuple:

$$G = (N, \Sigma, P, S)$$

where:

- N = a finite set of non-terminal symbols
- Σ = a finite set of terminal symbols
- P = a set of production rules of the form:

$A \to \gamma A$, where $A \in V, and\ \gamma \in (N \cup \Sigma)^*$

- $S \in N$ is the start symbol

It consists of a set of production rules that describe how strings in the language can be generated from a start symbol. Each production rule in a CFG has a single non-terminal symbol on the left-hand side and a string of terminals and/or non-terminals on the right-hand side. The key characteristic of a CFG is that the left-hand side of every production rule consists of only one non-terminal, meaning the rule can be applied regardless of the surrounding context in the string.

C. A context-sensitive grammar (CSG) consists of the following components

$$G = (N, \Sigma, P, S)$$

where:

- N = a finite set of non-terminal symbols
- Σ = a finite set of terminal symbols, with $N \cap \Sigma = \emptyset$
- P = a set of production rules of the form:
- $S \in N$ is the start symbol

The key restriction for production rules in a context-sensitive grammar is that each rule must be length-increasing, meaning the number of symbols on the left-hand side is less than or equal to the number on the right-hand side ($|\alpha| \leq |\beta|$ for each rule $\alpha \to \beta$) except for the special case where $S \to \epsilon$ is allowed only if ϵ (the empty string) is part of the language and S does not appear on the right-hand side of any rule. In CSGs, productions can include context around the non-terminal being replaced, allowing the rules to be sensitive to neighbouring symbols.

In the next section, the preliminaries is presented.

3. PRELIMINARIES

This section outlines key definitions related to formal language theory, including the concept of the *n*-th order limit language.

Definition 1: Alphabets, [1]

The alphabet, represented by the symbol, Σ, refers to a fundamental concept in formal language theory, serving as the primary set of symbols utilised to create strings in a language. It establishes the basis for defining the grammar rules as well as the syntax pertaining to a language.

Definition 2: Strings, [1]

A string represents a finite sequence of characters or symbols selected from a particular alphabet, expressed as |w|.

Definition 3: Language, [1]

L is expressed as a set of possibly finite sets of strings pertaining to some finite alphabet. Here, $\Sigma*$ denotes the set of all possible strings of a finite alphabet Σ. Note that, $L \subseteq \Sigma*$.

Definition 4: The *n*-th Order Limit Language from the Generation of Language Perspectives, [19]

Suppose $\gamma = (V, T, A, R)$ is an EH splicing system in which V is an alphabet of the system, $T \subseteq V$ denotes a terminal alphabet, $A \subseteq V^*$ represents a set of axioms, while R refers to a set of rules over A, where $R_n, 1 \leq n \leq k$. We define $L(\gamma) = \sigma^*(A) \cap T^* \neq Ln(\gamma) = \sigma^*(A) \cap T^*$ and L_n can be obtained in $L(\gamma)$ by removing string that can be spliced using rules.

Suppose $L(\gamma)$ represents the splicing language from the splicing system. Then, we express $L_n(\gamma)$ given that n represents the limit language order that is defined by the number of rules utilised in the splicing system. Thus, the rule must be different from each other, and the length of the rule must be the same. The language formed by splicing system is $L(\gamma) = \sigma^*(A) \cap T^*$. Therefore, $L_n(\gamma)$ is different from the set of string $L_{n+1}(\gamma), L_{n+2}(\gamma), \ldots$, provided that $\bigcap_{i=1}^{n} L_i$ with $L_1(\gamma) \not\subset L_2(\gamma) \not\subset \ldots \not\subset L_n(\gamma)$.

Then, the following section presents the past results where the computational power of *n*-th order limit languages were being increased to one level according to the Chomsky hierarchy.

4. RESULT

The generalization of the *n*-th order limit language is previously proven by the cases given in [20] and they are summarised as Table 1 b below.

Case	Order of the limit language	Language type [21]	Language type after increasing the computational power	Method
1	1st	Regular language	Context free	Regular expression > CFG > CFL
2	2nd	Context free	Context sensitive	Modify grammar from CFG to CSG
3	3rd	Context free	Context sensitive	Modify grammar from

				CFG to CSG
4	n^{th}	Context free	Context sensitive	Modify grammar from CFG to CSG
5	n^{th}	Context free	Context sensitive	Modify grammar from CFG to CSG
6	n^{th}	Context free	Context sensitive	Modify grammar from CFG to CSG

In this paper, the following theorems are discussed where, **Theorem 1**: Let S be a splicing system that generates a regular language where the splicing operation is limited to a single rule application (i.e., of order n=1). If the splicing system is modified to allow context-free grammar, then the resulting system can generate a language context-free language.
Proof: To increase the computational power from a regular language to a context-free language, first represent the splicing language using a regular expression. Then, convert the regular expression into a context-free grammar to generate the desired context-free language.

Case 1:
Let $\gamma = (\{a,b\},\{a,b\},\{ab\},\{a\#\ b\})$ be an extended H splicing system that contains one axiom and a rule. The first order limit language, $L_1(\gamma)$, produced is $L_1(\gamma) = \{aa, bb\}$ [21]. The language produced is regular because it results from regular grammar, as presented below.
The grammar $G_1 = (\{S,A,B\}\{a,b\},\{S\},P)$ which P given as $P = \{S \rightarrow a|Aa|b|Bb, A \rightarrow a\ and\ B \rightarrow b\}$. From here, the language generated by $L(G_1) = \{aa, bb\}$ is a regular language and also can represent as regular expression as (aa|bb) . The regular expression (aa|bb) describes a language containing only two strings: "aa" and "bb". The equivalent set of strings is: $\{aa, bb\}$.
Then, to increase the computational power up to context free language, we must convert the grammar into CFG. Let $G_1^* = (\{S\}\{a,b\},\{S\},P)$ which P given as $P = \{S \rightarrow aa\ and\ S \rightarrow bb\}$. From here, the language generated by $L(G_1^*) = \{aa, bb\}$. The grammar conforms to the format of a context-free grammar, where each production has a single non-terminal on the left-hand side. The language is finite and simple, making it expressible as a CFG. Then, the language generated by CFG is context free language.
We can see that the language produced is still the same but the grammar production, P , is different. The differences are given in the table as follows.

Table 1. Comparison of Grammar, G_1 and G_1^*.

Grammar	$G_1 = (\{S,A,B\}\{a,b\},\{S\},$	$G_1^* = (\{S\}\{a,b\},\{S\},$
Production in Grammar	$P = \begin{Bmatrix} S \rightarrow a\|Aa\|b\|Bb, \\ A \rightarrow a\ and \\ B \rightarrow b \end{Bmatrix}$	$P = \begin{Bmatrix} S \rightarrow aa\ and \\ S \rightarrow bb \end{Bmatrix}$
Type of Grammar that Generates the Language	Regular grammar	Context-free grammar
Language Produced	$\{aa, bb\}$	$\{aa, bb\}$
Type of Language	Regular language	Context-free language

Theorem 2: Let S be a splicing system that generates a context free language where the splicing operation is limited to a multiple rules application (i.e., of order n>2). If the splicing system is modified to allow context-sensitive grammar, then the resulting system can generate a language context-free language.
Proof: To increase the computational power from a context-free language to a context-sensitive language, the grammar must be modified from a context-free grammar (CFG) to a context-sensitive grammar (CSG). This modification allows the generation of context-sensitive languages. In [21], there is a theorem shows that all the *n*-th order limit language produced from the extended-H system using a finite number of initial strings, *m* and a finite set of rules, *n* will produce context-free language. Let Case 2 summarized the second order limit language until *n*-th order limit language.

Case 2
Let $\gamma = (\{c, x_1, d, g, x_2, h\},\{c, h\},\{cx_1dgx_2h\},\{c\#x_1d\$g\#x_2h\})$ be an extended H splicing system containing one axiom and two rules. As previously stated, for the generalization of an *n*-th order limit language, the case where two rules are used can also generalize the *n*-th order limit language. The generalization of the limit language, L_n , $L_n\ (\gamma) = \{ab^nc, ab^na, cb^nc\}$ [21].
The grammar $G_2 = (\{S,X,a,b,c\}\{a,b,c\},\{S\},P)$, where P is given by $P = \{S \rightarrow aXc|aXa|cXc|$ and $X \rightarrow b|\varepsilon\}$. From here, the language generated by grammar, $L(G_2) = \{ab^*c, ab^*a, cb^*c\}$, is a context-free language since it is generated by a regular and context-free grammar. The language can be summarise as $L_n = \{ab^nc \mid n \geq 1\}$. Since the language has been generalize into $L_n = \{ab^nc \mid n \geq 1\}$, the grammar also can be generalize into simplest grammar production. The grammar $G_n = (\{S,\}\{a,b,c\},\{S\},P)$, where P is given by $P = S \rightarrow aBc$, $B \rightarrow bB$ and $B \rightarrow \epsilon$.
Previously, we defined the *n*-th order limit language as $L_n = \{ab^nc \mid n \geq 1\}$, as our initial focus was solely on the combination of languages produced by limit languages. In this study, we aim to explore the types of languages based on their generative properties. To broaden our analysis, we also consider both left and right patterns. Accordingly, we modify the limit language to $L_n{}^* = \{a^nb^nc^n \mid n \geq 1\}$, while still preserving the underlying structure produced by the splicing system. This modification demonstrates that by adjusting the form of the generated language, it is possible to obtain a

context-sensitive language from the *n*-th order limit language.

Then, to increase the computational power up to context sensitive language, we must convert the grammar into CSG. Let $G_n^* = (\{S, A, B, C, X, Y, Z\}\{a, b, c\}, \{S\}, P)$ such that P is given as $P = S \rightarrow aSBC \mid abc, CB \rightarrow BC, aB \rightarrow ab,\ bB \rightarrow bb, bC \rightarrow bc \text{ and } cC \rightarrow cc$. From here, the language generated by ${L_n}^* = \{a^n b^n c^n \mid n \geq 1\}$. The grammar conforms to the format of a context-sensitive grammar and this pattern requires matching three symbol counts, a, b, c which cannot be done with a context-free grammar and it requires context-sensitive production rules. Hence, the context-sensitive grammar above is used to generate such strings and produce context sensitive language.

Apart from Case 1, we observe that the language is slightly altered, but we still consider it as *n*-th order limit language. The differences are given in the table as follows.

Table 2. Comparison of Grammar, G_n and G_n^*.

Grammar	$G_n = (\{S,\}\{a,b,c\},\{S$	$G_n^* = (\{S, A, B, C\}\{a,b,c\},$
Production in Grammar	$P = \begin{Bmatrix} S \rightarrow aBc, \\ B \rightarrow bB, \text{and} \\ B \rightarrow \epsilon \end{Bmatrix}$	$P = \begin{Bmatrix} S \rightarrow aSBC \mid abc, \\ CB \rightarrow BC, \\ aB \rightarrow ab, \\ bB \rightarrow bb, \\ bC \rightarrow bc \text{ and} \\ cC \rightarrow cc \end{Bmatrix}$
Type of Grammar that Generates the Language	Context-free grammar	Context-sensitive grammar
Language Produced	$\{ab^n c \mid n \geq 1\}$	$\{a^n b^n c^n \mid n \geq 1\}$
Type of Language	Context-free language	Context-sensitive language

5. CONCLUSION

The results of the two theorems demonstrate how the computational power of a splicing system can be progressively enhanced by modifying its underlying grammatical framework. Initially, a splicing system restricted to a single rule application (order n=1) and based on regular language principles can be extended by incorporating context-free grammar rules. This allows the system to generate context-free languages, which are capable of expressing recursive and nested structures by modifying the grammar.

Furthermore, by increasing the rule application (e.g., n>2), the system can model more complex dependencies such as modifying the grammar that go beyond the capabilities of context-free grammars. As a result, the language generated by the system becomes context-sensitive.

These findings highlight a hierarchical relationship which starting from regular languages, one can move to context-free, and then to context-sensitive languages by gradually increasing the structural complexity of the grammar and the operational power of the splicing system. This progression reflects the well-known Chomsky hierarchy and shows that splicing systems, when suitably enhanced, can simulate increasingly complex classes of formal languages.

6. ACKNOWLEDGEMENT

This research is fully funded by grant RDU223229. The authors would like to thank the Ministry of Higher Education for providing financial support under the Fundamental Research Grant Scheme (FRGS), FRGS/1/2022/STG06/UMP/02/4 (University reference RDU220131).

7. REFERENCES

[1] B. Steinberg, Languages and Automata. De Gruyter, 2024. doi: 10.1515/9783110984323.

[2] T. Head, “Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors,” Bull Math Biol, vol. 49, no. 6, pp. 737–759, 1987.

[3] T. E. Goode and E. Goode, “Splicing to the Limit,” Aspects of Molecular Computing, pp. 189–201, 2004.

[4] D. A. Simovici, Introduction to the Theory of Formal Languages. WORLD SCIENTIFIC, 2024. doi: 10.1142/13862.

[5] F. Starke, “Exploring the topological entropy of formal languages,” Theor Comput Sci, vol. 849, pp. 210–226, Jan. 2021, doi: 10.1016/j.tcs.2020.10.025.

[6] F. Karimi and S. Turaev, “Fuzzy Splicing System,” in 6th International Conference, ICCCI, 2014, pp. 20–29. doi: 10.1007/978-3-319-11289-3.

[7] M. P. Santono, M. Selvarajoo, W. H. Fong, and N. H. Sarmin, “Bounded-Addition Fuzzy Simple Splicing Systems,” J Algebr Stat, vol. 13, no. 2, pp. 2079–2089, 2022, [Online]. Available: https://publishoa.com

[8] M. P. Santono, M. Selvarajoo, W. H. Fong, and N. H. Sarmin, “The Properties of Bounded-Addition Fuzzy Semi-simple Splicing Systems,” in Proceedings of the International Conference on Mathematical Sciences and Statistics 2022 (ICMSS 2022), Atlantis Press International BV, 2023, pp. 354–363. doi: 10.2991/978-94-6463-014-5_31.

[9] A. Firdaus Yosman, W. Heng Fong, and H. Izzati Mat Hassim, “Characteristics of Formal Languages Generated by Bonded Systems Bonded Insertion and Deletion Systems View project Grammars Controlled by Petri Nets Under Parallel Firing Strategies View project Characteristics of Formal Languages Generated by Bonded Systems,” 2022. [Online]. Available: https://www.researchgate.net/publication/361575982

[10] S. Verlan, “Head Splicing System and Applications to Bio-informatic,” PhD Thesis, Université Paul Verlaine-Metz, 2004.

[11] Y. S. Gan, “Finite Automata In DNA Splicing And Sticker Systems With Weights,” Universiti Teknologi Malaysia, 2015.

[12] N. H. Sarmin, W. H. Fong, and S. Turaev, “The generative power of weighted one-sided and regular sticker systems,” no. June, 2014, doi: 10.1063/1.4882584.

[13] M. Selvarajoo et al., “Computational Power of Probabilistic Bidirectional Sticker System in DNA Computing On the structure of metacyclic-2 groups View project Computational Power of Probabilistic Bidirectional Sticker System in DNA Computing,” Online, 2015. [Online]. Available: www.ceser.in/ceserpwww.ceserp.com/cp-jourwww.ceserpublications.com
[14] M. Selvarajoo, F. Wan Heng, N. Haniza Sarmin, and S. Turaev, “Probabilistic semi-simple splicing system and its characteristics,” Jurnal Teknologi (Sciences and Engineering), vol. 62, no. 3, pp. 21–26, 2013, doi: 10.11113/jt.v62.1884.
[15] M. Selvarajoo, F. W. Heng, N. H. Sarmin, and S. Turaev, “The properties of semi-simple splicing system over alternating group, A3,” in Journal of Physics: Conference Series, IOP Publishing Ltd, Apr. 2021. doi: 10.1088/1742-6596/1770/1/012001.
[16] N. Chomsky, “On certain formal properties of grammars,” Information and Control., vol. (59), no. 90362–6, 1959, doi: 10.1016/S0019-9958(59)90362-6.
[17] W. T. Fitch, “Toward a computational framework for cognitive biology: Unifying approaches from cognitive neuroscience and comparative cognition,” Phys Life Rev, vol. 11, no. 3, pp. 329–364, Sep. 2014, doi: 10.1016/j.plrev.2014.04.005.
[18] N. Chomsky, “Simplicity and the form of grammars,” Journal of Language Modelling, vol. 9, 2021.
[19] S. H. Mohd Khairuddin, M. A. Ahmad, M. S. Mohamad, and R. Sokkalingam, “Redefining n-th Order Limit Languages in Extended-H Splicing System,” WSEAS TRANSACTIONS ON COMPUTER RESEARCH, vol. 13, pp. 58–64, Dec. 2025, doi: 10.37394/232018.2025.13.6.
[20] S. H. Mohd Khairuddin, M. A. Ahmad, and N. Adzhar, “An Introduction to n-th Order Limit Language,” Engineering Letters, vol. 30, no. 1, 2022.
[21] S. H. Mohd Khairuddin, M. A. Ahmad, and M. S. Mohamad, “Classification of n-th Order Limit Language in Formal Language Classes,” Journal of Advanced Research in Applied Sciences and Engineering Technology, 2023.