International Journal of Research in Circuits, Devices and Systems

E-ISSN: 2708-454X
P-ISSN: 2708-4531
IJRCDS 2021; 2(2): 71-76

www.circuitsjournal.com
Received: 17-05-2021
Accepted: 20-06-2021

Soham Mitra
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Correspondence
Soham Mitra
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Regular expressions: A detailed study for the understanding of their role and methods for efficient application

Soham Mitra

Abstract
Regular expressions (also known as "regex") are widely utilized for many practical applications and purposes, both in construction of Finite Automata and in programming languages. They are the constructs used to develop and build efficient systems of “Regular Languages”; however, they are fundamentally difficult to understand and re-use, resulting from a lack of abstraction methods, which inevitably causes Regular Expressions to grow very large in a short span of procedures. There are numerous regular expressions that are present through resources for the same problem going around, numerous of which usually are incorrect, resulting in it being harder to locate and utilize the appropriate regular expression for a given problem statement. Due to regular expressions being used widely in a variety of fields, low understandability and subsequently usability has become a critical software engineering issue. We will be discussing how Regular Expressions are indispensable in language engineering as well as in software engineering. We will cover how Regular Expressions provide a framework for construction of most “Regular Languages” which is one of its primary uses. We will be proposing a number of complementary representations not dependent on any other representation that might be used to explain regular expressions in this study. We also will be giving methods for computing those representations, as well as examples of how these approaches and the resulting explanations might be used in various contexts. Some of the representations aid in identification of errors in regular expressions, in addition to simplifying understanding [1]. Our examination reveals that our methods are broadly applicable and, as a result, can have a major impact on software engineering practice and applications. We thus establish how Regular Expressions come into play in theoretical as well as application-based studies and establish methods to reduce errors in usage of Regular Expressions.

Keywords: Regular expressions, regex, computation, software engineering, automata

Introduction
Regular Expressions are a type of language-defining notations used extensively in computation as a basic construct for developing regular languages. Regular Expressions are closely intertwined with Finite Automata, an essential component of computation, and can be used to describe or construct them [1]. We require an introduction to computation. Information technology has maintained a critical role in the scale of the world for the past 25-30 years. But we yet do not have a proper grasp of what this technology allows us. For some reason, people continue to believe that the concept of computation refers to numbers, punctuation, or acronyms [2].

Despite the actual fact that the concept of computation isn't limited to numbers or punctuation marks, the meaning of this term remains unclear. Almost everyone within the world believes that if someone works in accounting, he or she must be conducting calculations. Many of us, however, disagree over whether the brain could be a computer or something else. If the brain could be a computer, all thoughts prove to be calculations, and every computation end up to be thoughts. The virtual world in computer games is additionally known to be a mirrored image of the real world that has evolved as a result of computations [2].

Regular expressions are the source of a limited but very useful language for describing a wide range of formatting and small text-based languages [1]. Regular expressions developed within the framework of formal language theory, and one in every of its commonest use has been as a part of compiler scanners. However, they now have uses that go far beyond them. Regular expressions, for instance, are employed in editors to change structured text [3].

They're quite often used to analyze network protocols [4] and specify events in distributed systems [5]. Regular expressions are used for finding of viruses via signature scanning [6], web mining [7] and as XML data alternatives [8].
Finite Automata are one of the most indispensable tools in Computational studies. They are used to make abstract machines for development and verification of regular languages using computational models. They provide methods of describing finite state computations like matching strings for generation of a regular language. However, the concepts and constructs of Deterministic or Non-Deterministic Finite Automata may be hard to understand for people not familiar with the concepts. Thus, Regular Expressions also act as a semantical translation of such Automata using variety of symbols, allowing them to be implemented by a wide range of users.
However Regular Expressions are not without their fair share of problems. Three major problems in regular expressions are their Complexity, Errors, Version Proliferation. Lack of abstraction leads to one of the primary sources of problems regarding regular expressions [1]. They tend to grow very large and thus introduce a scope of being incorrect, as it is difficult to verify large regular expressions. Also, modifications have a high chance of introducing new errors leading to lack of reusability, as they sometimes exceed even 100 to 500 characters. On this basis, we aim to establish methods for explaining regular expressions which will aid in-depth understanding of the structure of regular expressions, and can assist users to be able to search repositories with higher efficiency and identify fallacies in regular expressions encountered.

1.1 Organization of work
Section 2: Discuses the role of Regular Expressions in Regular Languages and how they are interconnected with Finite Automata, providing a symbolical understanding of the State Diagrams of Finite Automata through inter conversions, the established methods for which have been discussed.
Section 3: Gives us an understanding of the reasons that debilitate widespread usage of Regular Expressions and lists the problems that exist in them, and further, provide solutions for the same.

2. Regular expressions and finite automata
Regular Expressions have the other side of the coin as Finite Automata. One is a structural representation; another is a semantic representation. As it stands, Regular Expressions and Finite Automata are interchangeable and one can be converted to the other and back. And there are many usable tools for the same.

2.1 Finite automata to regular expressions
Oftentimes there is a requirement of converting Finite State Automata constructs into Regular Expressions to algebraically describe the strings that the automata can accept. They offer a tool to simplistically define Regular Languages without the necessity of elaborate construction of state diagrams and transition tables. They even enable users without detailed knowledge of regular languages a method to efficiently apply the pattern matching capabilities provided by Regular Languages by easy-to-understand symbolic notations.
Few classical algorithms are:
- Algorithm based on Arden's lemma [10, 28].
- The McNaughton-Yamada algorithm [11].
- The state elimination technique [12].

Arden's lemma
Arden's lemma is an algebraic approach with an algorithm for solving the problem of transformation of finite automata to regular expressions [10]. It proposes a set of equations dependent on languages for a specific finite automaton. The ith equation in this case is used to describe X_i, which is the set of words w that allow the automaton in question to undergo a transition starting at ith state to one of the existing accepting states by scanning w. The resulting system of equations can be solved employing a method equivalent to Gaussian elimination that assists in removing the indeterminate states such as X_i.

Lemma 1: Let Σ be an alphabet, and let K, L ⊆ Σ∗, where K does not contain the empty word ε. Then the set K∗L is the unique solution to the language equation X = K ·X +L, where X is the indeterminate.

McNaughton-Yamada algorithm
This technique is a general-purpose algorithm for extracting state graphs from regular expressions. When using this technique, fewer states than "2P+1" are often required, and the working time of this method is limited to "(m|x|)". (m = (s2)-the maximum number of links in the state graph. In regular expressions, s denotes the number of alphabetic letters. x denotes a bit vector) [24].
This system utilizes a matrix containing the regular expression entries. If n is used to denote the number of states in the given automaton A, the looping algorithm employs a ranking on the collection of states, running in n iterations. The entry ajk in the matrix (ajk) j, k calculated in round 1 is an result that defines the non-empty words w of calculations of A which start with j and end at k, in such a way that any state that exists in between of the computation is at a lower rank than i. After this, it is not a difficult task to form the corresponding regular expression describing L(A).

State elimination algorithm
It works by using an extended finite automaton, which consists of transitions described by regular expressions and not alphabetical characters. An NFA A's processing can be explained for simplicity as scanning the word given as input, on each of its letters, non-deterministically transitioning every state of the automaton with each letter of the word such that it is does not invalidate its transition table δ. When we read a word w ∈ Σ, it is visible that A transitions from jth to kth state if there exists an operation on word w taken as input, that transitions A from state j to state k. Similarly, we say that A is also set to move on input w from state j through U to state k for a subset U of the automaton A's state set S if there exist a computation applicable on the input word w which will take A and transition from state j to k without passing through any intermediate states exclusive of j and k [12].

Theorem 1: Let $n \geq 1$ and A be a n-state NFA over alphabet Σ. Then alphabetic width $|\Sigma| \cdot 4^n$ is sufficient for a regular expression describing L(A). Such an expression can be constructed by state elimination.

2.2 Regular expressions to finite automata
Sometimes we might have the requirement of transforming

Regular Expressions back into Finite Automata. The methods for the same are dependent upon the final automata to be achieved, namely ε-NFA, NFA or DFA. There are two such methods:

- Thompson's Construction [13].
- Construction of Glushkov automaton [14, 11].
- Chang and Paige's Algorithm [25].

Thompson's construction

Thompson's construction was made popular through the UNIX tool grep [13]. It is the recursive coupling of sub-automata through ε-transitions. This method divides regular expressions into smaller regular expressions, converts them to nondeterministic finite automata and then combines these automata to generate the first given regular expressions converted into NFA. Furthermore, with this technique, each letter in the regular expressions is taken and tested in the order in which they appear and then transferred to the nondeterministic finite automata, eliminating mistakes that may occur in the connections between the states and the route path. These smaller automata are linked side by side for union, and one after the other for concatenation, and iteratively for the Kleene star. This leads to generation of an ε-NFA consisting of finite number of states and transitions. [20, 21] provide a characterization of structures of the Thompson automaton in terms of the proposed combination of two words. Thompson's traditional construction underwent various levels of adaptation and optimization.

Construction of Glushkov automaton

Using the subset creation procedure, this approach transforms regular expressions to its respective nondeterministic finite automata. The states contained by this NFA instinctively convert to alphabetic symbols or, specifically, places in the regular expression between successive alphabets. To be more explicit, imagine r is a regular expression with a length of n alphabetic characters. In r, we use subscripts to refer to each letter's location (counted from left to right) in the alphabet. This produces a complementary expression r' with separate input characters over Σ' containing all letters in r'. To keep things simple, we'll suppose that the same notation is used for marking and unmarking, i.e., $r'' = r$.

Theorem 2: The Glushkov automaton M_E can be computed from a regular expression E in time linear in size (E) + size (M_E).

Chang and Paige's Algorithm

The transition from regular expressions to NFA is generated in the same asymptotic time as Berry and Sethi's algorithm [26]-"Θ(m)". In spite of that, it is a method that has enhanced the auxiliary memory-the quantity of "Θ(s)" kept in Berry and Sethi's [26] approach's memory. (Sdenotes the number of alphabetic letters in the regular expressions). The number of links existing between states in automation is lacking in McNaughton and Yamada's method. Because, considering in the worst-case scenario, the number of connections is "m = $\Theta(s^2)$".

3. Problems in regular expressions

3.1 Deficiencies existing in regular expressions

Lack of abstraction

Lack of abstraction mechanisms prevailing in majority of regular expressions, which in turn results in following issues:

- **Scalability:** Regular expressions tend to grow large within few methods, resulting in slight issues in comprehensibility of said regular expressions. Users are under the compulsion of taking them from other sources to describe the same sub expression in multiple instances, due to the lack of a naming procedure severely limiting the possibility of changing the size of regular expressions.
- **Lack of structure:** Even multiple occurrences of smaller expressions in same words as before cannot be removed from consideration via naming and must be copied. Repetitions such as these are troublesome to detect, but understanding of such shared structure is crucial to the process of understanding the meaning behind the regular expression in question. As a result, regular expressions are inefficient regarding informing users what shared features and lack of consistencies they contain.
- **Redundancy:** Multiple occurrences of sub expressions does not just obscure the structure and leads to increment of the size, but it also leads to creation of useless similar parts in the representations, which is prone to update anomalies during process of conversion of the regular expressions. This acts as a cause of numerous regular expression flaws and causes significant impact on future maintenance capability of the expressions.
- **Unclear intent:** Because of the fore mentioned issues, it is extremely difficult to effectively determine the ideal scope of a regular expression from how it is structured. Lacking such knowledge, it becomes bothersome to determine the accuracy of a regular expression, making it difficult to determine if a certain regular expression should be utilized or not. Furthermore, the challenge in interpreting a regular expression's aim turns it into an incredibly difficult task to choose from a repository of regular expressions and select the most efficient one for some specific usage [1].

Inability to exploit domain knowledge

To organize abstract conceptual realms, indirect mappings from domains which are present directly in experience are utilized [16]. For the first time, children learning arithmetic frequently rely on transferring the abstract realm of numbers to their digits [17]. If users are given a mapping to a less difficult to comprehend or more known area, the abstract notions become easier to grasp. An issue with regular expressions exists owning to it being a formal notation with no map being closed [18] to the field to which the user intends to utilize it in, which turns it bothersome for new users attempting to develop skills in regular expressions [19]. Furthermore, there exists no clear conceptual model for the expression evaluator's response.

Role expressiveness

An expressive system of notations must give users with different visual indications about the uses of its constituents [20]. Simple regular expressions contain a small number of guidelines for assisting the user in identifying the parts of the regular expression that choose the relevant regions of the input string. Users often maneuver around this by utilizing divided expressions that match to main areas of the input and indentation to visually identify the functions of sub expressions.

Error proneness
Regular expressions tend to undergo errors because there is no transparent distinction among the elements of the regular expression which are shared across the selectable variations and those that are not, and so contribute more to the variants that can be matched.

3.2 Explanations for error elimination in regular expression

In order to comprehend a particular Regular Expression in detail, it is indispensable to correctly identify its structure and goal of its contents. On extension to that, we utilize the structures of existing concept that a specific regular expression covers by offering symbol-based elaborations that can carry various symbolic constraints in a brief, high-level manner [1].

Structural Analysis and Decomposition
Regular expressions of bigger size are made up of smaller divided expressions that are frequently highly similar to one another. We can develop a representation that directly displays commonalities by automatically detecting and abstracting common subexpressions. Our approach is a twist on a very typical procedure for factoring common subexpressions. The process starts by converting as many continuous sequences of characters as possible that are void of brackets like (...), [...], or | into bracket containing forms. If the expression does not contain |, this step is not required. It does not lose the meaning of the regular expression owing to the fact that brackets do not modify the meaning of a regular expression on their own. The following step is to define names for sequences (without space specifications) that are understood as common things that appear in multiple instances in the expression. We utilize a number of methods that are possible to be learnt by oneself in the extraction of subexpressions to generate names for the divided expressions because the selection of names has a significant impact on the comprehensibility of the broken-down expression [22, 23]. Taking an example we show, a simple name indicates the regular expression that matches the name, such as img = *img*. Because many regular expression applications require the description of keywords whose capitalization does not matter, we also employ the convention that underlined names indicate regular expressions that match any capitalization of that term, such as a = [aA] or img = [iI][mM][gG]. In the example, frequent expressions are phrases that represent upper- or lowercase letters, such as [aA], and are therefore substituted by their names.

Format analysis
The majority of non-trivial regular expressions employ a variety of symbols and punctuations to separate various portions of the expression. A fixed derived sequence of characters that is part of any string represented by a regular expression is referred to as a format. Because the format is present in all strings, it adds nothing to the information expressed. Similarly, the variable part of a regular expression is referred to as its data part [1]. Individual strings of a format must be maximum by definition, which means that in the primary regular expression, there will always exist an instance of a (possibly empty) data string among two format strings. The condition at the start and end of a format is without restrictions; which means, a format may or may not begin or conclude the regular expression.

Analysis by user requirement
Analysis of a number of repositories have indicated that many regular expressions share common subexpressions. They occasionally have identical copies of the same subexpression, while other times the phrases are extremely close. This implies that there is a significant demand for, as well as a potential for, the repeated usage of regular expressions. In particular, the shared features may be used in couple of methods. Foremost, if a user wishes to develop a unique regular expression for a specific usage, it may be possible that certain elements of said expression are pre-existent in a repository as a smaller expression. The issue is determining the manner to locate them. If the names employed in the breakdowns generated as part of explanations were sufficiently detailed, these domain-specific names might be applied to find in or explore repositories. When explanations with detailed names have started to being stored in existing repositories, this will work considerably well in further uses for a long time.

4. Related works

This paper provides a new theoretical insight on regular expressions which can aid in increased usage of them in respective fields. This paper not only defines the interrelation between Regular Expressions and Finite State Automata, it provides methods for interconversion, unlike [24] where we only have tools to convert Regular Expressions to Finite Automata and [27] which has only tools for conversion of Regular Expressions to Finite Automata. Also, in contrast to how [1] handles methods of efficient and error-free usage of Regular Expressions, this paper first establishes a base of Regular Expressions so readers can better comprehend the solutions provided. The sections 2.1, 2.2 give a comprehensive guide to established methods of interconversion. Further, students, researchers are suggested to go through research work of Tyagi *et al*., [29-26] to know about Emerging Technologies and their importance (including issues and challenges) in the near future.

5. Conclusion and Future work

Our discussion led us through a discourse of how regular expressions are an indispensable tool in study of Computational theory. In our Introduction we discuss the requirement and scope of this research papers and introduces the broad concepts involved in writing the paper. We discuss several methods available to our disposal that facilitate conversion of given finite automata such as ε-NFA, NFA and even DFA to Regular Expressions in section 2.1. When we need to visualize exactly what languages our automaton may accept, which might not be clear directly through the state diagram. We also discussed in section 2.2 what methods are crucial in conversion procedure of a Regular Expression back into a Finite Automata in case where designers need to analyze the processing states involved in the regular expression during accepting each word of the language accepted by the regular expression. In section 3, we delve into research on what are the prevalent problems and restrictions that hinder widespread and flawless usage of Regular Expressions in their different functionalities and fields. In section 3.1 we finally discuss the major deficiencies existing in the system of regular expressions that lead to reduced usability, and provide error

elimination solutions for the same in section 3.2. In future there may be more established algorithms proposed with a more technical scope.

5. References

1. Erwig M, Gopinath R. Explanations for Regular Expressions. In: De Lara J, Zisman A. (eds.) Fundamental Approaches to Software Engineering. FASE. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, 7212. https://doi.org/10.1007/978-3-642-28872-2_27
2. Hopcroft JE, Motwani R, Ullman JD. Introduction to Automata Theory, Languages, and Computation, 3rd Ed, Prentice Hall, 2006.
3. Horswill Ian What is computation? XRDS: Crossroads, The ACM Magazine for Students. 2012;18:8-14. 10.1145/2090276.2090283.
4. Pike R. Structural Regular Expressions. In: EUUG Spring Conf., 1987, 21-28.
5. Wondracek G, Comparetti PM, Kruegel C, Kirda E. Automatic Network Protocol Analysis. In: Annual Network and Distributed System Security Symp. (NDSS 08), 2008.
6. Nakata A, Higashino T, Taniguchi K. Protocol synthesis from context-free processes using event structures. In: Int. Conf. on Real-Time Computing Systems and Applications, 1998, 173-180.
7. Lockwood JW, Moscola J, Kulig M, Reddick D, Brooks T. Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware. In: In Military and Aerospace Programmable Logic Device (MAPLD), 2003, 10.
8. Hur J, Schuyler AD, States DJ, Feldman EL. Sci. Miner: web-based literature mining tool for target identification and functional enrichment analysis. Bioinformatics (Oxford, England). 2009 Mar;25(6):838-840.
9. Hosoya H, Vouillon J, Pierce BC. Regular Expression Types for XML. In: In Proc. of the International Conf. on Functional Programming (ICFP), 2000, 11-22.
10. Arden DN. Delayed-Logic and Finite-State Machines. In T. Mott, editor: Proceedings of the 1st and 2nd Annual Symposium on Switching Theory and Logical Design, American Institute of Electrical Engineers, New York, Detroit, Michigan, USA, 1961, 133-151. Doi: 10.1109/FOCS.1961.13
11. Robert McNaughton, Hisao Yamada. Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers EC. 1960;9(1):39-47. Doi: 10.1109/TEC.1960.5221603.
12. Brzozowski JA, McCluskey EJ. Signal flow graph techniques for sequential circuit state diagrams. IEEE Transactions on Computers C-12(2), 1963, 67-76. Doi: 10.1109/PGEC.1963.263416.
13. Thompson K. Regular Expression Search Algorithm. Communications of the ACM. 1968;11(6):419-422. Doi: 10.1145/363347.363387.
14. Glushkov VM. The abstract theory of automata. Russian Mathematics Surveys 16, 1961, 1-53. Doi: 10.1070/RM1961v016n05ABEH004112.
15. Gruber Hermann, Holzer Markus. From Finite Automata to Regular Expressions and Back-A Summary on Descriptional Complexity. Electronic Proceedings in Theoretical Computer Science, 2014, 151. 10.4204/EPTCS.151.2.
16. Boroditsky L. Metaphoric structuring: understanding time through spatial metaphors. Cognition. 2000;75(1):1-28.
17. Curcio F, Robbins O, Ela SS. The Role of Body Parts and Readiness in Acquisition of Number Conservation. Child Development. 1971 Nov;42:1641-1646.
18. Blackwell AF, Green TR. Notational Systems-The Cognitive Dimensions of Notations Framework. HCI Models, Theories and Frameworks: Toward and Interdisciplinary Science, 2003, 103-133.
19. Blackwell AF. See What You Need: Helping End-users to Build Abstractions. J Visual Languages and Computing. 2001;12(5):475-499.
20. Giammarresi D, Ponty JL, Wood D, Ziadi D. A Characterization of Thompson Digraphs. Discrete Applied Mathematics. 2004;134(1-3):317-337. Doi: 10.1016/S0166-218X(03)00299-3.
21. Giammarresi D, Pony JL, Wood D. Thompson Languages. In Karhum¨aki J, Maurer H, P˘aun G & Rozenberg G. editors: Jewels are Forever: Contributions on Theoretical Computer Science in Honor of Arto Salomaa, Springer, 1999, 16-24. Doi: 10.1007/978-3-642-60207-8_2.
22. Bransford JD, Johnson MK. Contextual prerequisites for understanding: Some investigations of comprehension and recall. J Verbal Learning and Verbal Behavior. 1972;11(6):717-726.
23. Derek MJ. The New C Standard: An Economic and Cultural Commentary. Addison-Wesley Professional, 2003.
24. Batar M, Birant K. Development of an Efficient Tool to Convert Regular Expressions to NFA. Journal of Emerging Computer Technologies. 2021;1(2):38-43.
25. Chang C. From regular expressions to DFA's using compressed NFA's. Ph.D. Thesis, New York University, New York, 1992.
26. Berry G, Sethi R. From regular expressions to deterministic automata. Theoretical Computer Science. 1986;48(1):117-126.
27. Alih Akpa, Daniel Alih. Conversion of Deterministic and Non-Deterministic Finite Automata to Regular Expression using Brzozowski Algebraic Method. International Journal of Scientific and Engineering Research. 2020;11:53-60.
28. Conway JH. Regular Algebra and Finite Machines. Chapman and Hall, 1971.
29. Nair MM, Tyagi AK, Sreenath N. The Future with Industry 4.0 at the Core of Society 5.0: Open Issues, Future Opportunities and Challenges, 2021 International Conference on Computer Communication and Informatics (ICCCI), 2021, 1-7. Doi: 10.1109/ICCCI50826.2021.9402498.
30. Tyagi AK, Fernandez TF, Mishra S, Kumari S. Intelligent Automation Systems at the Core of Industry 4.0. In: Abraham A, Piuri V, Gandhi N, Siarry P, Kaklauskas A, Madureira A. (eds.) Intelligent Systems Design and Applications. ISDA 2020. Advances in Intelligent Systems and Computing, Springer, Cham., 2021, 13-51. https://doi.org/10.1007/978-3-030-71187-0_1.
31. Goyal Deepti, Tyagi Amit. A Look at Top 35 Problems in the Computer Science Field for the Next Decade, 2020. 10.1201/9781003052098-40.
32. Varsha R, Nair SM, Tyagi AK, Aswathy SU, Radha Krishnan R. The Future with Advanced Analytics: A

Sequential Analysis of the Disruptive Technology's Scope. In: Abraham A, Hanne T, Castillo O, Gandhi N, Nogueira Rios T, Hong TP. (eds.) Hybrid Intelligent Systems. HIS 2020. Advances in Intelligent Systems and Computing, Springer, Cham., 2021, 13-75. https://doi.org/10.1007/978-3-030-73050-5_56.
33. Nair Meghna Manoj, Tyagi Amit Kumar. Privacy: History, Statistics, Policy, Laws, Preservation and Threat Analysis, Journal of Information Assurance & Security. 2021;16(1):24-34, 11.
34. Tyagi Amit Kumar, Nair Meghna Manoj, Niladhuri, Sreenath, Abraham Ajith. Security, Privacy Research issues in Various Computing Platforms: A Survey and the Road Ahead, Journal of Information Assurance & Security. 2020; 15(1):1-16, 16.
35. Madhav AVS, Tyagi AK. The World with Future Technologies (Post-COVID-19): Open Issues, Challenges and the Road Ahead. In: Tyagi AK, Abraham A, Kaklauskas A. (eds.) Intelligent Interactive Multimedia Systems for e-Healthcare Applications. Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-6542-4_22.
36. Amit Kumar Tyagi, Aghila G. A Wide Scale Survey on Botnet, International Journal of Computer Applications. 2011 Nov;34(9):9-22. (ISSN: 0975-8887).