International Journal of Research in Circuits, Devices and Systems

E-ISSN: 2708-454X
P-ISSN: 2708-4531
IJRCDS 2022; 3(1): 05-12

www.circuitsjournal.com
Received: 20-10-2021
Accepted: 05-12-2021

C Lanston Davis
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Correspondence
C Lanston Davis
School of Computing Science and Engineering, Vellore Institute of Technology, Chennai, Tamil Nadu, India

Applications of finite automata towards string acceptance and lexical analysis

C Lanston Davis

Abstract
In this paper, we would like to discuss about the Applications of Finite Automata towards String Acceptance and Lexical Analysis. It was in 1943 when Warren McCulloch and Walter Pitts contented their idea (to create a human approach process) along with some neurologists, mathematicians and by some the first computer scientists. A lot more information was gathered, relating to Finite automata, Pushdown Automata, Turing machine etc. Finite Automata is basically a compiler where it constructs itself to provide its application in phases such as Lexical Analyzers, Synthetic Analyzers, Code Generation, Intermediate Code Generation, Code Optimization etc. which can be helpful for many basic machines such as text editors spell cheekers, for designing sequential circuits too. Lexical analyzers are very much useful in computing complex sequence of characters into tokens where the tokens have a definite set of meaning which could be decoded by the computer. As said earlier, the analysis helps in the division of the program to be compiled into separate tokens, then it helps in detecting the error tokens and aids the user by directing out the line and array of error (points out the row and column number of errors precisely), it removes the unnecessary comments and spaces too. So, in this research paper we will inform you more in detail about its applications which will further deal with string acceptance, and what and how are necessary for the above-mentioned functions, and discuss in detail how Finite Automata which clearly has no space or said to have "Finite Space" for a limited time period would help to do such tasks as compilers.

Keywords: Finite automata, pushdown automata, turing machine, spell-checkers, text editors, tokens

1. Introduction
Psychology and trait loving researchers have acknowledged the social media as a center of knowledge, a researcher from the University of Cambridge is able to conclude that people are able to create friends, who have nearly equivalent behaviors. The behaviors were studied through a Facebook application: My Personality, "Words are always said to convey feeling and emotions, and sometimes confusion", the key meaning is that they are said to contain useful information of the user which the media person or some other person would like to examine. it is through these words; the behaviors are analyzed. Mostly through the comments sent, the posts they like (from the comment section), which are further more analyzed through its captions, their tags, their mutual friend's data, etc. In a jist, a new language handling procedure has been introduced to counter or to keep in check or otherwise to understand the people's perception on various topics posted through articles, research papers, blogs, elections, etc. But its significance and impact is felt when its machine-based algorithms is run for sub-applications such as Plagiarism Detection, Bioinformatics and DNA sequencing, Digital Forensics, Spelling checker, Spam filters, Search engines or content search in large databases, Intrusion Detection System [1] etc. which we will deal in the fore-coming part of the paper. Sentiment Analysis basically uses Lexical Analysis and String acceptance to mine the emotion of the user, but it entirely isn't that easy, it has a lot of drawbacks which are explained in sources such as "Hasinai Rahmath P "Opinion Mining and Sentiment Analysis-Challenges and Applications", clearing the air for this statement is that, unlike organized sentence framing which can be seen in newspaper printing or document reading, otherwise research paper making, the social platforms such as Twitter use informal sentence making, usage of idioms, tedious expressions, all turn over the perspective knowledge of the analyzer, hence to deal with the arduous languages, in this paper I would explain the applications where Finite Automata would exercise Lexical Analysis and String Acceptance consolidating and building the role of sentiment analysis much more defined and better.

Furthermore, as this paper will deal with Applications, we would look on an application of the header based on Compilers too, for an overall understanding [14].

2. A preview on string acceptance

A lot of algorithms are continuously being proposed to the land of computation and development, the time when computer technology stated to rise to its hill peak of knowledge, compilers were being developed for all the applications based on programming and web development came into picture, it was the growth subject of the Theory of Computation. Hence, researchers and innovators put in everything they had in order to save the complexity of the machine.

In order to better it in many ways, a lot of people put together many interesting algorithms such as an algorithm to avoid the numerous comparisons (reduces time), an algorithm known as the shift algorithm where the right shift up to one digit is carried out continuously, where obviously mismatch operations occur all the time, but at one point of time, at one point of the shift an even match algorithm successful is obtained, this algorithm is known as the Naives approach.

In the field of Multiple Pattern Matching Algorithms, Kleene in the year of 1956, proved the similarity and Boolean 1 of equivalence between regular expressions and finite automation, which we have come a long way till now, but, in those periods, it was an invention to solve the string-matching problem. As said earlier about Brute Force algorithm, it was proposed by Morris and Pratt in the year of 1970.

Just so we make ourselves clear, there are four types of string matching as sub-classifications for the higher-level matching classifications, which work at the lower level such as left to right matching, right to left matching, specific order matching and no order matching.

Continuing the Brute Force, this pattern matching checks and pre-processes the pattern by aligning the characters left to right and then it compares the string with it (it captures the idea when and not to left shift by one by thinking of the given pattern), now just for information, this algorithm isn't a best case because it is used to skip some mid-way characters on pre-processing phase in every loop.

An update of the Brute Force algorithm is the KMP (Knuth Morris Pratt) algorithm, a set of updates of improvements to the Pratt and Morris algorithm declined in a change for a significant time complexity rather it proposed a better searching performance than the latter one, this was achieved in 1970.

A further derivation using the best time, performance, results obtained from the Naïve and the KMP algorithm was brought into picture by Boyer and Moore in the year of 1977, this ALGO had run from the last character i.e.: right to left categorization.

And one of the important Algorithms based on the character comparison is the usage of the Trie data structure, which stores and retrieves data based on the storage of keys in the rules of a BST method.

Another category of lexical and string matching is the DFA method, it follows an important algorithm when it comes into perfection with profession, i.e., Automation-Matcher algorithm, its automata range from the first state to its last with every single one letter or character is proposed, maybe not one but state elements are made to fulfil the pattern sequence.

The past decades viewed in a diction of algorithm relating to Multiple Pattern String Matching such as in the algorithm proposed by Alfred V. Aho and Margaret J. Corasick which helps in the construction of automata for language and patterns recognized by the machine in the pre-processing phase just line in the brute force algorithm. Based on the algorithms such as the Aho-Corasick and the Boyer-Moore, Commentz Walter's derived algorithm helped in the search of multiple patterns, and Rabin Karp's algorithm too used the same ideology of searching.

In concern to all these data, the basement of the automation got stronger and healthier, with these as the foundation, many more applications in essence with the lexical and string acceptance have been brought forth as mentioned, we will look more in on the functioning of those individual applications in the further data of the research.

3. Construction of lexical analyzer

The first stage of the compiler is known as the Lexical Analysis, its process wholly depends on the analyzer which is fundamental in identifying and scanning over the language which is comprised of a valid string-token. The indexes of the string are further broken down and scanned which is then compared to whitespaces, run for comments etc. which is then removed, and sent to the syntax analyzer for further processing, in case if any invalid token happens to be scanned by, then it generates an error (see Figure 1).

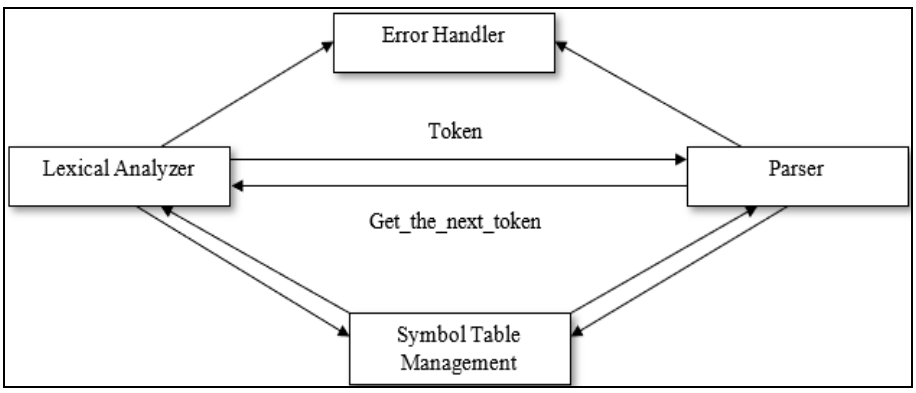

Fig 1: Construction of Lexical Analyzer

Lexical Analyzer are excellent function performance machines that work as scanners, there are constructed either by establishing a connection with the scanners. For a little language it is linked together, however for dialects with

huge syntax ought to be decoupled in light of the fact that methods utilized in these Lexical Analyzers is the most basic machine function in accordance with the Chomsky table of Computation- Deterministic Finite Automata. Stage disentanglement and can be improved to bigger degree can be accomplished through divide and ruling them, basically because of its distinction in methods and techniques, used in the respective stages. Additionally, the language's string set might shift from one machine to other which we can restrict it to scanner itself as opposed to taking forward to additional stage. These DFA performers are additionally liable for scanning and pulling out unwanted characters or special ones on the grounds that syntax with void area is truly challenging to compose and execute also. Parsers and scanners can be maker buyer pair where the maker function is performed by the scanner (produces tokens) and the purchasing function is performed by the parser which screens and splits the tokens and the string built by the scanner. With regards to execution, switch case is considered the most ideal way, why. Maybe because it can pick any type of token instead of having recursive functions carry them out [1].

4. Applications

4.1 Spell checkers

With basic knowledge we understand that this cannot be done using a random machine learning algorithm unless we have a memory allotted to it, that exactly why, there is a need for a basic data structure before it uses the Finite Automation to search a pattern of them, so the data structure used here is a Trie. The trie is built with a pre-defines, organized set of languages and pattern in each branch of it, then the Pattern is searched the occurrence by reaching to its maximum level, in essence, the final state of the pattern run by the finite automation, refer Figure 2 [2, 3].

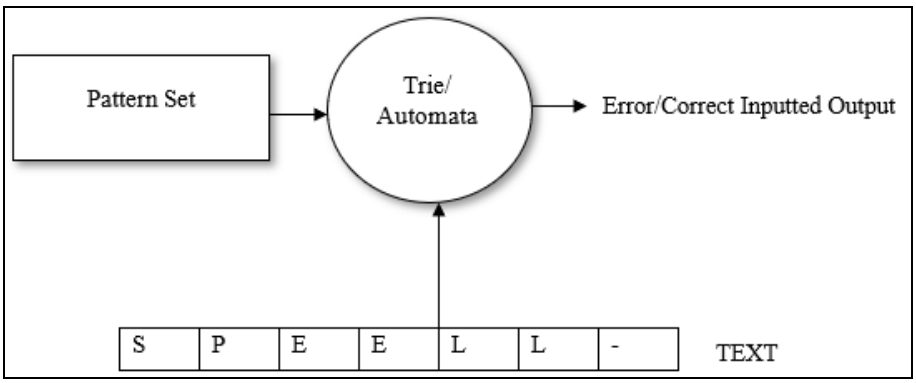

Fig 2: Spell Checkers

4.2 Spam filters

Popularly known as the Spam Detection Systems, the principle used by these machines is solely adept to string matching, spams are any kind of data that enters into the users account, mostly unsolicited and spontaneous, they maybe in the medium of emails, messages, phone calls too, most importantly they result in the financial pocket money losses, hence it is a need to discard spams and this is accomplished by string matching, the matching is done by suspecting signature patterns into consideration, refer Figure 3 [3].

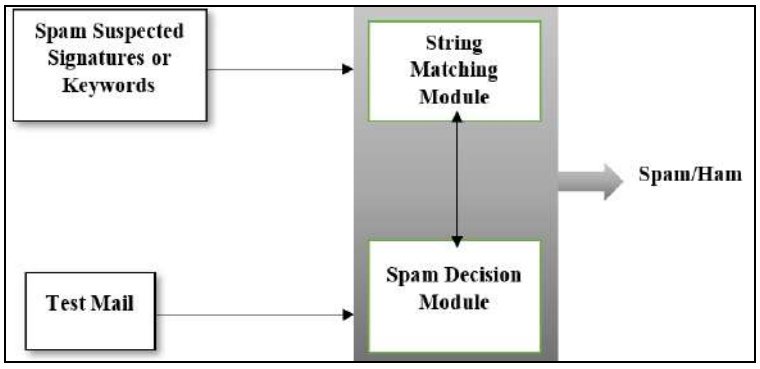

Fig 3: Spam Detection Systems

4.3 Intrusion detection system

Anti-virus application may have a tint of a database of computer scrutinizing code inside of them, after all, this detection system examines the incoming data packets one by one, one way to detect whether the code contains malicious code or not is by comparing the very data within the received data packet with the malevolent code stored inside our own device databases for comparison, so thinking about a successful comparison method would definitely lead to string acceptance, here the pattern matching would be the code to be exercised for comparison. And once if a match if confirmed then, the alarm is generated whether or not to exhibit the application or document, otherwise the incoming packet into our system or not i.e., an alarm, refer Figure 4 [3, 24, 25].

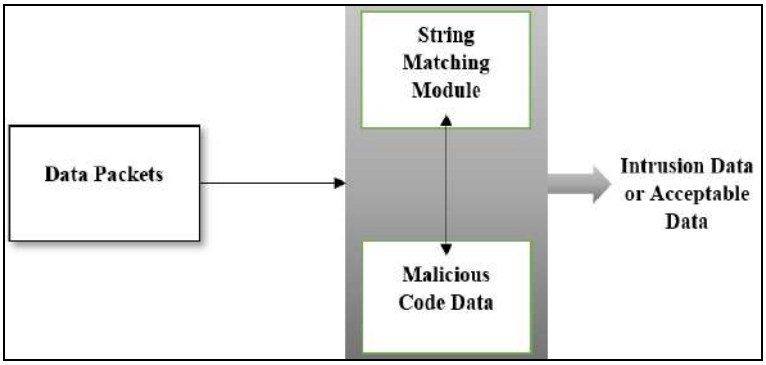

Fig 4: Intrusion Detection System

4.4 Search engines

The essence in search of a keyword deployed to the Google or Bing or Quora, anything as such in the internet is available to us with the help of a search entry, this search entry is run through the databases, and the suitable and most compact article is received back to the user's device, again this is successfully executed using the string acceptance. The search entry transmitted is processed by the search engine where the organizing of data and text suits the most efficient pattern or language in header, or the content lines of an article, where it is then further displayed, like in Google the search text is entered, then the text is run by the search engine which then searches the articles simultaneously using its most suitable algorithm, but it does not matter, the pattern is searched in multiple articles at the same time then the application of google finally sorts them according to the header containing the search text and furthermore, the article with a lot number of search texts is then further displayed in the sorted order, Figure 5 [3].

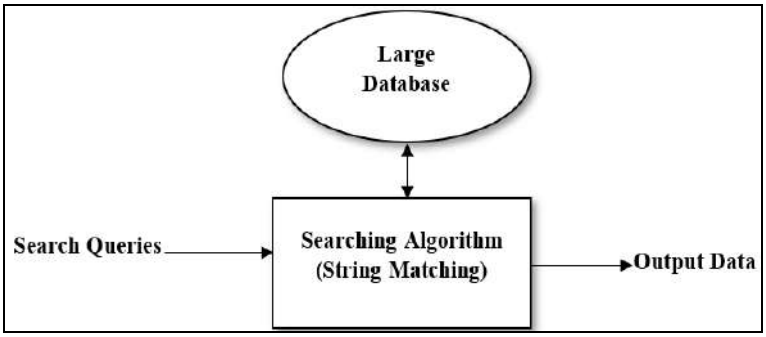

Fig 5: Search Engine

4.5 Plagiarism detection

One of the easiest applications to understand, where comparison is the only choice to prove exact copying is committed or not, this takes a huge finite automata pattern comparison from each from one article to another, where string acceptance is carried out where we can compare text and detect the similarities between them. Hence, we can conclude whether it is an original test or not, Figure 6 [3].

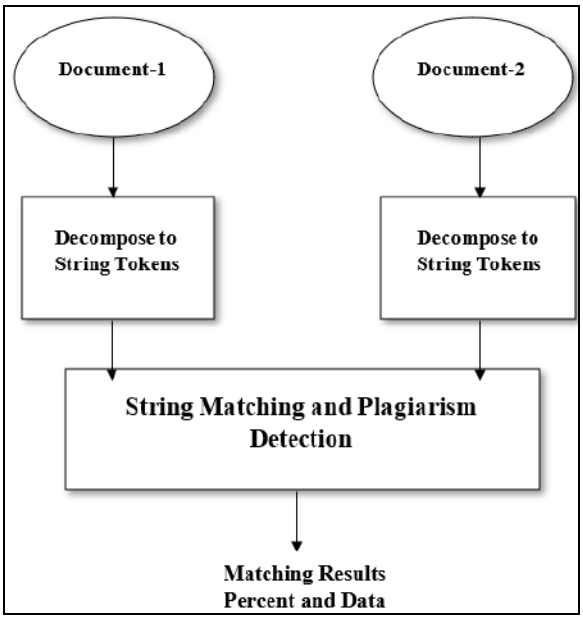

Fig 6: Plagiarism Detection

4.6 Bioinformatics

A product of Computer Science in relating to Biology and Information Technology to act a solution to modern day biological problems is where we need to Bioinformatics, as explained before comparison of genetic sequences in the languages of binary or DNA analyzing will to require the best complexity algorithms and effectiveness and overall trust and methods to carry out these sensitive applications, hence the string-matching modules are necessary, Figure 7 [3].

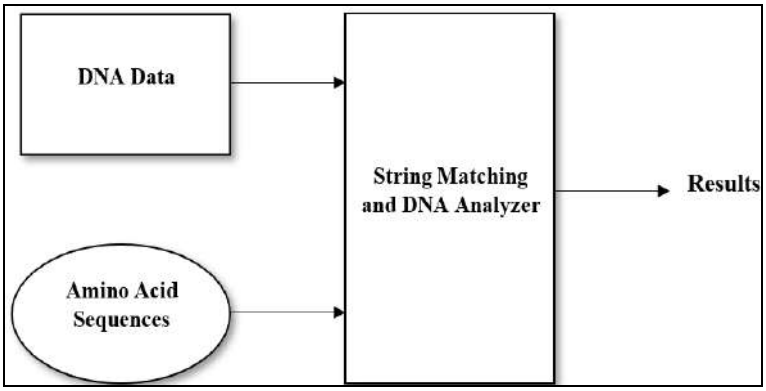

Fig 7: FA Use in Bioinformatics

4.7 Digital Forensics

investigation of devices found at crime site, where the investigation carries out the examination of every single byte of information run continuously searching for patterns relating to eccentric keywords or any other information or data, running through every single byte can be externally compared with the search engine or the bioinformatics examples, but this is in regard to detail extraction in another perspective than the former examples. Here the text mining of data is extracted and processed from loads of sites and applications etc. but this can be made easier by further adding in string acceptance where we can diagnose out pattern matching in terms of topic tracking, answering of questions, basically information extraction to a very readable extent, Figure 8 [3].

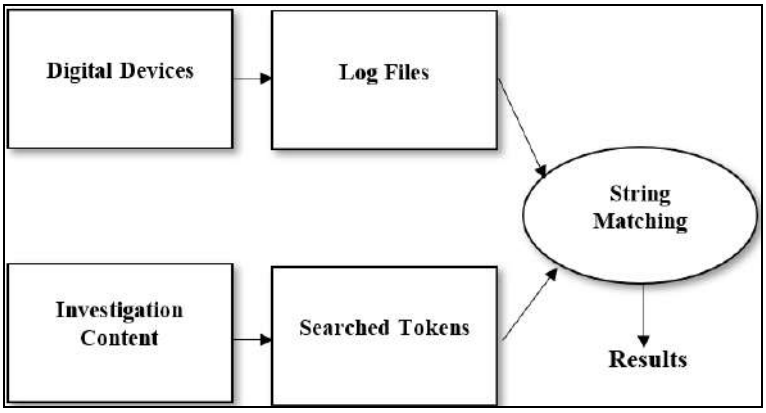

Fig 8: FA Use in Digital Forensics

5. Understanding GNU C compiler in correlation to lexical analysis

5.1 Before we begin, a small recap on

Lexical Analysis enacts as the first phase in the process of compilations and it sums up very important applications. Functionalities as a first phase include:-

A. Input to the lexical analyzer is the source code itself and the analyzer is gravely guided to the interpretation of the lexical units present in the code by reading it character by character, hence this analyzer is also referred to as the scanner.
B. The identified lexical units (tokens) are thereby passed to the next phase, i.e.: the syntax phase via a table, namely the symbol table where the tokens collected are stored after the scanning process.
C. The limit of action exercised by lexical analysis extends not only within the domain of the existing code, rather it can extend to the linked files too.
D. The scanner is also accountable for eradicating tabs, blanks, comments and newlines which accounts for the scanning of the source program.
E. Lexical analyzer stands out in the detection and upholding the track line of errors that occur within the source code, no matter which phase the source code undergoes after lexical analysis.

5.2 Briefing on action

String Acceptance, a very close entity when compared to one such application, namely: Program result checking, an efficient method to resolve the problem of cost, for a verification of a formal software, or to be precise, in this topic one will understand the significance and use of lexical analysis and string acceptance by understanding their application in verifying the results of a compiler or a software or some similar application, From here on, I would like to focus on understanding the applications by using the GCC compiler.

5.3 Understanding the compiler

In brief, we will understand the description and the processing of Programming languages inside a compiler. Generally, each compiler has a frontend and a backend, in this discussion, we will confine towards the frontend, which will deal with the lexical analysis as one of the frontend checkers. Well, as said earlier a compiler has a frontend and a backend, the frontend will deal with decoding or otherwise to be precise, checking the input program of the program language, the frontend processes them and sends them to the backend for optimization of the unique solutions gathered from the frontend to convert them into a machine code language for further processing, but what if, there are several unique solutions for a piece of input, now this is a serious compile compiler error, to solve this we need to indulge a little deep into the reading of input, i.e., we need more checker processors. Hence a new method was devised namely, the Program Checking with Certificates, where the compiler produces a certificate on how it has computed its result and thereby, now the certificate is passed to the checker where the checker analyzes the result once again by running the processes in its perspective with reference to the same steps as from the certificate hence at present there is a three-stages process check for checking a GNU C Compiler. First, the regular expression casts a role by separating several individual characters from input, and these processes are implemented easily using the finite automata, and using lexical analysis, this ends the first stage. The second-stage comprises of converting the obtained scanner tokens into a unique syntax-tree whose traversal is designed in index which can be obtained by running the tree, a top-down traversal; and in the final stage, attributes are associated along with the abstract tree. Hence to brief, these very phases require the correct token sequence, and syntax tree with accurate finite automation, similarly the attributes are to be computed fittingly for the own good of context-free grammar and attribute grammar.

5.4 Checker functions based on lexical analysis

The conversion of input characters to tokens is performed by the scanner (Lexical Analysis). Assigning of respective string tokens into their respective tables as in the symbol table is the function of lexical analysis, but this isn't easy, the scanner pointer would have to discard all unnecessary characters, as in whitespaces, comments, line breaks etc. The inputs would contain a lot of type of tokens, they may be in terms of identifiers, or command lines etc., the assigning of the value token and stuff would be stored to the symbol table where the symbol and the value would be present. To avoid indeterminism, the lexical analysis is mapped using regular languages which is surely governed by finite automata and convened with the rule of the longest pattern e.g., the sequence 'f', 'o', 'r' is mapped to token of FOR not to an identifier.

The checker now using the program checking certificate as its memory now recomputes and checks whether the computed sequence obtained is according to the transition rules of the finite automation or not. And for information, the checking perspective also plays a vital role for this approach to obtain success, in other words, the top to down approach (the character sequence) is only to be followed as said earlier the computation is unique, the doubts and problems for non-uniqueness arises from wrong indexing of the whitespaces, comments and line breaks, which is different from the check certificate first obtained. For example, if the string sequences for the tokens 'WHILE' ('w', 'h', 'i', 'l', 'e') and 'IDENT' (an identifier) should be separated. The longest pattern rule would be applied if their respective character sequences isn't separated, recognizing that, concatenation is performed on only one identifier of the character consisted sequence of the 'WHILE' token from the 'IDENT' token. This is a completely different approach, when compared to the Scanning Algorithm (original), and I believe, the effort the compiler puts in to accept tokens with uniqueness will not go in vain. And when time comes to recompute the part, where Lexical analysis performs its role, we bring in another application i.e., an interactive theorem prover which has a program extraction facility (e.g., Isabelle or HOL). The key feature of this extraction facility is the direct obtaining of the Machine Learning code which we try to generate: to finally deliver the code it to the Backend. We may think why is it necessary two times but as said earlier a unique, no erroneous perfect code and some more reasons where we will further understand from the rest of the paper hence leaving the last reasons are our main objectives to be obtained from the Frontend. To summarize, source program, computed token sequence from the Lexical analysis (Finite Automation) and specification are the three inputs and the data is only sent to the Backend if and only if the recomputed sequence from the ITP (Interactive Theorem Prover) is same as the derived one.

5.5 GCC functions based on lexical analysis

The above processes vary a little in terms of GCC (Abbr. of GNU C Compiler), lets understand this better beginning with a simple example which only comprises of Identifiers and Numbers; 'a', 'b' and '0', '1' respectively. Now this as basic information, let the GCC allow input of any list of characters of any language pertaining to criteria, here let's consider a sequence of numbers and a sequence of characters. The scanner in the ITP i.e., Isabelle or HOL provers take input and process them to finally result whether the final outputs are an 'IDENTIFIER', In case of a sequence of characters or a sequence of number: 'INTEGER'. But the most interesting fact is that, the scanner from the lexical analysis can compute more in detail information that the provers. But for a fact, the successor phases after the provers would necessarily need more than just the data type for computing, they also need the value of the token obtained, in order to do so, four parameters are setup are the legs for the primitive recursive function based on a scanner, they include, the token-list and their corresponding values, the present-existing state, the order and sequence of characters that has not been read yet, the string sequence of characters that represent the value of the current token.

We were able to bring froth the properties of the scanners spoken in different definitions and perspectives through this GCC, first, we were able to bring forth a scanner where which relies on the additional input which was able to plot does the state transitions and its activities carried out by its primitive recursive function and similarly using the same function, we were able to process a token obtained from a string based on the present setup of the finite automata.

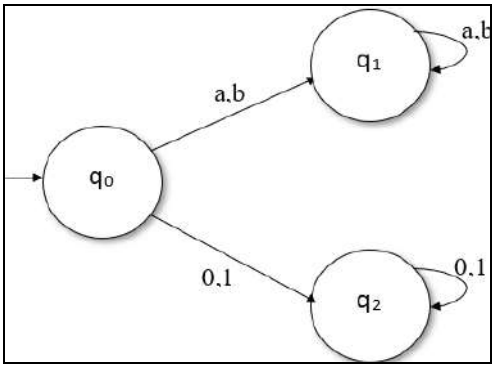

Fig 9: GCC Functions

5.6 Encompassing conclusion on GCC

Just as explained before, but the process for the conditions in the GCC is to be detailed a little further; we can classify this application of Lexical Analysis to a total of three steps, Derivation of the Tokens, Determination of the Finite Automata and the Specification in Interactive theorem Prover (Isabelle or HOL).

5.6.1 Derivation of the tokens

A GNU C Compiler does not have the knowledge to differentiate the pre-processing tokens and tokens, rather for the succeeding compilation phases, it is only said to use the pre-processed computed tokens, unlike the C standard where it is able to compute both the type of tokens.

5.6.2 Determination of finite automation

The outlook and periodic assessment of the tokens in the C standard would suffice to determine the Finite Automata.

5.6.3 Specification in interactive theorem prover

This is explained earlier, the two types of scanners is used to define the lexical analysis in Isabelle and HOL [15]. Further, some other uses of emerging technologies and raised issues in the same for the smart era can be found in [22-29].

6. Conclusion

In the rising world where accomplishments matter, going to the moon, beginning of earth, a pollution free life, and mostly in a social life where more than 90% percent of humans strive about very much wish for a simple user-friendly society, realizing maybe another perspective, but simple things make life simple for an individual, in terms of money and time. The computer industry has grown around this area striving to make products more user friendly, theory of computation and especially Finite Automata, have shown significant contributions to serve the most simple but complicated applications and overall understanding for a better, progressive life, for not only the people, but for the big tented companies who show-off their progressions. Through this project I understood the milli needs and application of this Automata, an idea within a outlook of a simple shell, is so useful, this proved its contributions as a pave way for a simple society and for progressive technology, for industries of education, compilers, patents, medicine, websites etc. Most importantly, its significance, which I hoped to understand from the beginning of the project.

7. References

1. An Exploration on Lexical Analysis Farhanaaz Assistant Professor Department of Computer Science Aditya Institute of Management Studies and Research Bangalore, India https://www.researchgate.net/publication/311251505_An_exploration_on_lexical_analysis_
2. Finite-State Spell-Checking with Weighted Language and Error Models-Building and Evaluating Spell-Checkers with Wikipedia as Corpus Tommi A Pirinen, Krister Linden´ University of Helsinki-Department of Modern Languages Unioninkatu 40 A-FI-00014 Helsingin yliopisto https://www.researchgate.net/publication/252066408_Finite-state_spell-checking_with_weighted_language_and_error_models.
3. Kapil Kumar Soni, Rohit Vyas, Amit Sinhal TIT College Bhopal, Madhya Pradesh India https://www.researchgate.net/publication/304305210_Importance_of_String_Matching_in_Real_World_Problems
4. https://www.geeksforgeeks.org/applications-of-string-matching-algorithms/
5. https://er.yuvayana.org/finite-automata-example-equivalence-limitation-and-application-of-fa/
6. 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
7. 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
8. https://www.geeksforgeeks.org/introduction-of-lexical-analysis/
9. http://www.ijcset.com/docs/IJCSET15-06-05-037.pdf
10. https://nptel.ac.in/content/storage2/courses/106108113/module2/Lecture2.pdf
11. https://www.brainkart.com/article/Lexical-Analysis_8074/
12. A Lexical Analyzer with Ambiguity Support Luis Quesada, Fernando Berzal and Francisco J. Cortijo Department of Computer Science and Artificial Intelligence, CITIC, University of Granada, 18071 Granada, Spain, https://www.scitepress.org/Papers/2011/34768/34768.pdf
13. Social Media Based Opinion Mining Using Lexical Sentiment Analysis 1 2 3 4 5 Puja Munjal, Aditi Gupta, Mahima Abrol, Hema Banati and Sandeep Kumar 1, 2, 3 Jagannath International Management School, GGSIPU, Vasant Kunj, Delhi 4 Dyal Singh College, University of Delhi, Delhi 5 Department of Computer Science and Engineering, Jagannath University, Jaipur (PDF) Social media Based Opinion Mining Using Lexical Sentiment Analysis (researchgate.net)
14. Luis Quesada, Fernando Berzal and Francisco J. Cortijo Department of Computer Science and Artificial Intelligence, CITIC, University of Granada, 18071

Granada, Spain https://www.scitepress.org/Papers/2011/34768/34768.pdf
15. Sabine Glesnera Simone Forstera Matthias J¨agera a Fakult¨at f¨ur Informatik, Universit¨at Karlsruhe, 76128 Karlsruhe, Germany Email: {glesner|simone|matthias}@ipd.info.uni-karlsruhe.de, https://www.researchgate.net/publication/220370238_A Program_Result_Checker_for_the_Lexical_Analysis_of _the_GNU_C_Compiler.
16. Ezhilarasu P Associate Professor Department of Computer Science and Engineering Hindusthan College of Engineering and Technology Coimbatore, India. prof.p.ezhilarasu@gmail.com Krishnaraj N Head of the Department of Information Technology Sree Sastha Institute of Engineering and Technology, Chennai, India drnkrishnaraj@gmail.com. http://www.ijcset.com/docs/IJCSET15-06-05-037.pdf
17. Dan Chalmers, Simon Fleming, Ian Wakeman and Des Watson Informatics, University of Sussex, Brighton, U.K. Email: D.Chalmers@sussex.ac.uk, https://www.researchgate.net/publication/220876256_Rhythms_in_Twitter
18. Praveen Saini and Renu Sharma Department of Computer Science & Engineering, Amrapali Institute of Technology & Sciences, Haldwani (U.K.), https://www.researchtrend.net/ijet/pdf/87-F-769.pdf.
19. https://www.guru99.com/compiler-design-lexical-analysis.html
20. https://www.tutorialspoint.com/compiler_design/compiler_design_regular_expressions.htm
21. Latha L, Preethi M, Grahalakshmi R. International Journal of Engineering and Advanced Technology (IJEAT), 2019 Sep, 8(6S3). ISSN: 2249–8958. https://www.ijeat.org/wp-content/uploads/papers/v8i6S3/F12050986S319.pdf
22. 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.
23. 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.
24. Goyal Deepti, Tyagi Amit. A Look at Top 35 Problems in the Computer Science Field for the Next Decade, 2020. 10.1201/9781003052098-40.
25. Rekha S Malik, Tyagi AK, Nair MM. Intrusion Detection in Cyber Security: Role of Machine Learning and Data Mining in Cyber Security, Advances in Science, Technology and Engineering Systems Journal. 2020; 5(3):72-81.
26. Mishra S, Tyagi AK. Intrusion Detection in Internet of Things (IoTs) Based Applications using Blockchain Technolgy, 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2019, 123-128. Doi: 10.1109/I-SMAC47947.2019.9032557.
27. 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.
28. 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.
29. 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).