Chapter 3 Language Recognition through Context-Free Grammars and Natural Language Processing

Capítulo 3 Reconocimiento de Lenguajes mediante Gramáticas Libres de Contexto y Procesamiento del Lenguaje Natural

CABALLERO-HERNANDEZ, Hector†*, MUÑOZ-JIMÉNEZ, Vianney and RAMOS-CORCHADO Marco A.

Universidad Autónoma del Estado de México, Faculty of Engineering
Tecnológico de Estudios Superiores de Jocotitlán, División de Ingeniería en Sistemas Computacionales.

ID 1st Author: *Hector, Caballero-Hernandez* / **ORC ID:** 0000-0002-2790-833X, **CVU CONAHCYT ID:** 445998

ID 1st Co-author: *Vianney, Muñoz-Jiménez* / **ORC ID**: 0000-0003-2180-6743, **CVU CONAHCYT ID:** 44343

ID 2nd Co-author: *Marco A., Ramos-Corchado* / **CVU CONAHCYT ID**: 37345

DOI: 10.35429/H.2023.13.25.39

H. Caballero, V. Muñoz and M. Ramos

*hcaballeroh045@alumno.uaemex.mx

Abstract

At present, language recognition systems have great relevance, because these systems allow to identify if the lexical units belong to a certain language, this type of tools have allowed to generate translators, word identifiers, and recently sentiment analyzers have been built, thanks to current advances it has been possible to achieve with natural language processing the analysis of sentences developed by human beings to be interpreted by computerized systems. This research shows the application of techniques for recognizing the belonging of strings to formal languages using context-free grammars, on the other hand, a software has been developed for the recognition of feelings in text. The results achieved indicate a recognition of 100% of the analyzed strings, as well as the interpretation of the analyzed sentences, performing the coding of a parser and a sentiment analyzer.

Context, Sentiment, Recognition, Analyzers

Resumen

En la actualidad los sistemas de reconocimiento de idiomas tienen gran relevancia, debido a que permiten identificar si las unidades léxicas pertenecen a un determinado idioma, este tipo de herramientas han permitido generar traductores, identificadores de palabras y recientemente se han construido analizadores de sentimiento, gracias a los avances actuales. se ha podido lograr con el procesamiento del lenguaje natural el análisis de oraciones desarrolladas por seres humanos para ser interpretadas por sistemas computarizados. Esta investigación muestra la aplicación de técnicas para el reconocimiento de la pertenencia de cadenas a lenguajes formales utilizando gramáticas libres de contexto, por otro lado, se ha desarrollado un software para el reconocimiento de sentimientos en un texto. Los resultados obtenidos indican un reconocimiento del 100% de las cadenas analizadas, así como la interpretación de las frases analizadas, realizando la codificación de un analizador sintáctico y de sentimiento.

Procesamiento del lenguaje natural, Gramáticas, Reconocimiento

Introduction

Language is one of the most powerful tools that human beings must transmit information and thus achieve different objectives (Denning, 1978). With the development of computers, language research became important, to generate programming languages that would speed up the process of execution of computer programs. Currently, new tools have been developed that take advantage of the creation of lexical and syntactic analyzers to guide them to the interpretation of the language of the human being and to be able to carry out the construction of the meaning of the sentences that are emitted digital networks, as well as the generation of automated responses by digital assistants.

Because a large part of the knowledge that currently exists is on the internet, whether in text or video format, in volumes of great concentration, artificial intelligence has been harnessed for information processing through machine learning, big data, visual recognition, as well as natural language processing, this last area specializes in probabilistic analysis, ambiguities as well as the extraction of information that is present in text and to be able to generate discourses, natural language responses between machines and people (Chowdhary & Chowdhary, 2020).

In computer theory a formal language is a set of strings of finite length symbols formed from a finite alphabet (**Σ**), has a series of rules, with which an explanation or meaning (Hopcroft, Motwani, & Ullman, 2001). The empty set is represented by Æ, and the set formed by the empty string, represented by the symbol Î, are languages. Formal languages can be specified as:

– Strings produced by a formal grammar (Chomsky hierarchy).
– Strings produced by a regular expression.
– Chains accepted by an automaton, (as an example you have the Turing machine).

On the other hand, a Turing machine is a formal model, which has a finite control, an input tape that is divided into cells, and a tape head that sweeps one cell from the tape at a time (Hopcroft, Motwani & Ullman, 2001).

The structure of languages can be described through grammars. One type of grammar is context-free grammar (FCG), which is a set of variables also known as syntactic nonterminal categories each of which represents a language (Vayadande *et al.*, 2023). Languages that are represented by variables that describe recursively in terms of the same variables and primitive symbols called terminals. The rules of grammar that relate to variables are known as production rules. A common production states that the language associated with a given variable contains strings that are formed by concatenation of strings taken from languages represented by other variables (Sipser, 1996).

The symbol "=>" denotes the act of derivation, which is understood as the substitution of a variable for the right side of a production. A context-free grammar is defined as.

$$G = (V, T, P, S) \tag{1}$$

Where:
V is a set of variables
T is a set of terminals
P is a finite set of productions of the form A=> α, where A is a variable and α is a string of symbols taken from (V È T).
S is the initial symbol.

Therefore, L is called a Free Context Language (CFL) if it is L (G) for some CFG. A string of terminals and α variables is known as a sentence form if S=> α. The L (G) language is the set of all the words that belong to an alphabet (Qureshi *et al.*, 2023). The alphabet is that element that contains the set of terminals (Xing *et al.*, 2009).

The Chomsky *Normal Form* or CNF is described as a context-free language without Î, it is generated by a grammar in which all productions are of the form A = > BC or A = > a. Here A, B and C are variable and α is a terminal (Kosheleva & Kreinovich, 2023). Grammars allow to pose the rules of a language because it describes the phases for the realization of some process or several subprocesses of an entity or phenomenon to be characterized.

Note that regular grammars restrict the rules that contain on the left to a non-terminal and on the right a single terminal, usually followed by a non-terminal. The rule $S \rightarrow \varepsilon$ is allowed if *S* does not appear to the right of any rule (Chen *et al.*, 2020).

The implementation of grammars in computer software requires the development of a lexical analyzer (scanner) and a parser. The lexical analyzer is a program that analyzes a certain language and produces as output a series of *tokens* or symbols. Symbols are used in the parser stage. A parser is responsible for converting the strings or tokens to a data structure, in this way the data organized to later generate an analysis of the previously treated code and make the compilation process possible. The final product is the analysis tree.

The descending parsers are what build the syntactic tree of the statement to be recognized, starting with the initial symbol or root, until reaching the terminal symbols that form the statement. Lexical and parsers are implemented in the compiler, which is software dedicated to the translation of one language to another, which can be interpreted by a computer, to execute the instructions that have been written. Particularly context-free grammars are used to identify the primordial elements and proceed to check the belonging of sentences to a language (Lasser *et al.*, 2019).

With the development of artificial intelligence (AI) it has been possible to solve tasks, which were particularly intended to be solved by expert systems, which were limited to evolve to adapt to new solutions. AI is a discipline related to the theory of computation whose goal is to emulate some of the human intellectual faculties in artificial systems. Human intelligence refers to sensory perception processes (vision, hearing, taste, among others) and their consequent pattern recognition processes, so the most common applications of AI are data processing and system identification. The most frequent applications of AI include fields such as robotics, image analysis or automatic word processing (Russell & Norving, 2004).

Artificial intelligence has different branches, of which the following stand out.

a) Fuzzy logic. It is a method based on the reasoning of logical expressions that describe the memberships to fuzzy sets. The important concepts are the fuzzy sets, responsible for interpreting the predicate of a set that has no bounded boundaries. One of the disadvantages of fuzzy sets is that the rules are defined from the construction of the model.

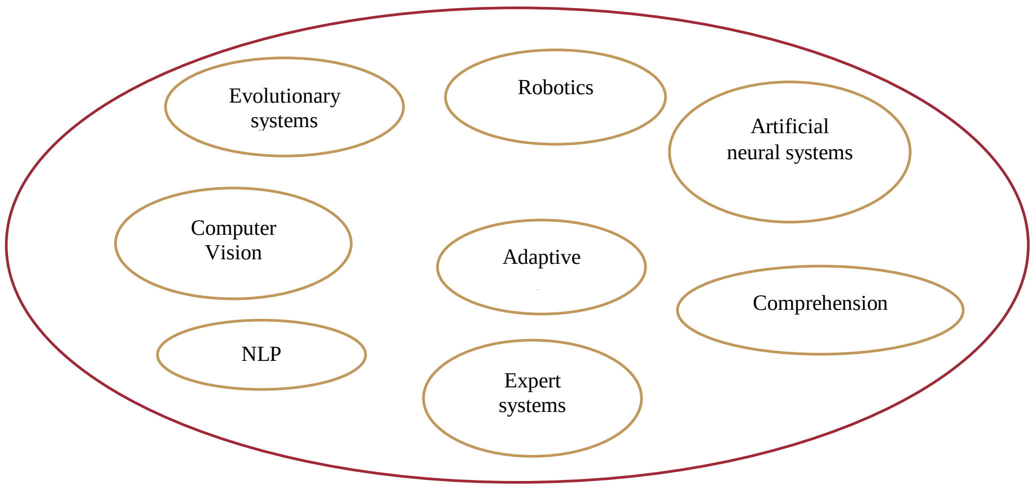
b) Genetic algorithms. Represent a stochastic search in which successor states are generated by combining two parent states, genetic algorithms are created with sets of randomly generated states, which are named population, individuals are represented with a string over a finite alphabet. The population in the GA evolves in a different way than it would in a real situation, because the model does not take other variables that exist in a real environment.

c) Neural networks. Neural network models are based on the functioning of biological brains, and have gained skills to develop distributed computing, digital image recognition, among others.

d) Expert systems. Expert systems are programs based on a knowledge base to make inferences and solve problems of high complexity that are traditionally solved by human beings (Brock & Grad, 2022). An expert system is a computational system capable of emulating the decisions of an expert human (Giarratano, 2001).

Some areas of artificial intelligence are contained in Figure 1.

Figure 1 Areas of artificial intelligence



Source: Giarratano (2001)

Due to the complexity involved in natural language recognition, there are different methods used by NLP for this (Nadkarni, 2011). These methods are listed below.

Support vector machines (SVM). SVMs are responsible for classifying the entries that are defined as words to categorize them, using mathematical transformations. Generally, the most used functions are Gaussian functions, to form a series of subsets of data that will be used for training (Galindo *et al*., 2020), (Murillo-Castañeda, 2021).

Hidden Markov models (HMM). The systems where a variable can change state, and a series of passible outputs are generated. The sets of possible states and unique symbols are finite and known. These models are used for speech recognition, where the waveform of a spoken word is matched to the sequence of individual phonemes (Ching, 2006), (Norris, 2011).

Other models used in NPL are CRFs, which are responsible for generalizing logistic regression to sequential data in a similar way to HMMs. The models are used to predict state variables based on the observed variables. As an example, there is the moment to write or pronounce the distinction that a person has, by which the model would interpret that it must follow the name of a subject (Nadkarni *et al*., 2011).

Computer systems have had a profound evolution, because, from the first information systems to the development of artificial intelligence, it has allowed them to acquire new capabilities for solving the problems of everyday life. Evolutionary algorithms have proven their efficiency for numerical analysis (Neri & Tirronen, 2010), (Qin *et al*., 2008), (Qin & Suganthan, 2005), (Dragoi *et al*., 2013).

The systems that implement evolutionary algorithms have the characteristic of entering information through pure text, audio, or video, then this information is analyzed and filtered for analysis, all the information is stored to function as the memory of the system, this memory can change position to adapt to the resolution of problems.

Literature review

The developments on string recognition mechanisms belonging to formal languages have been widely studied and there are different applications for language validation in various areas as can be seen Aschermann *et al*. (2019) for the verification of errors in languages, while in M. Ganardi *et al.* (2021) they presented the development of grammars to achieve balance in programs (Numaya *et al.*, 2023). Some other applications have focused on the use of grammars for biological applications as in (Huang *et al*., 2019) for RNA strand modeling.

In other works, such as Torr *et a*l. (2019) they relied on extracting knowledge from the definitions of synthesis problems to guide the construction of the grammar used by Grammatical Evolution and complement its adequacy function to improve the precision in the set of synthesis problems of reference programs in the field. In research such as Hemberg *et al.* (2019) and Shin *et al*. (2020) they focused their efforts on studying the evolution of grammars in computational environments, as well as on language recognition.

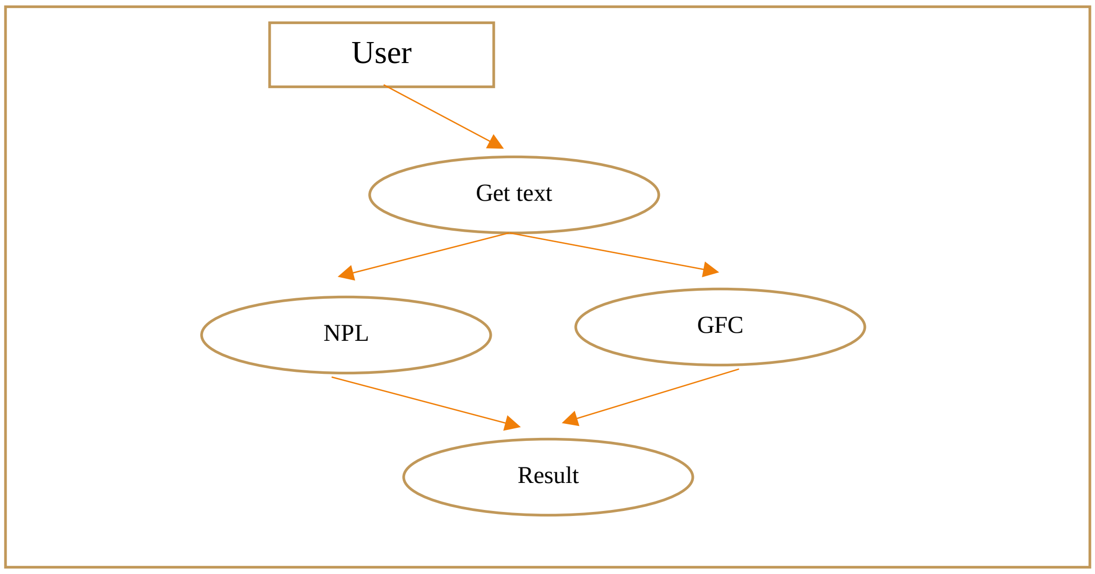
Due to the important advances that have been achieved in formal languages, it has been possible to accommodate the development of new tools that can analyze information in social networks Zucco *et al*. (2020), Can & Alatas (2019). In research such as that of Kauffmann *et al*. (2020) they have focused on developing a framework for the analysis of feelings with commercial uses, because commerce in social networks is of high importance, on the other hand, in Himelboim *et al*. (2020) they deal with problems the analysis of feelings in social networks such as Twitter through data cluster. Sentiment analysis on the internet is extensive, due to the study of mental health conditions Htet *et al*. (2019), commercial applications Khrais (2020) and in the field of education Xiao *et al*. (2020).

As can be seen, computational linguistics and natural language processing, present large areas of application, due to their extensive management of information, allowing the generation of new languages for computer, with tools that accelerate the coding process, as well as commercial, biological, and educational applications, being one of the tools with greater in computing and informatics.

Proposed Scheme

The present proposal consists of integrating a software with the ability to analyze the input strings of a language which is analyzed using context-free grammars, as well as the ability to interpret the natural language of a person to perform a sentiment analysis on the input strings. Figure 2 shows the general structure of the string analysis system.

Figure 2 General diagram of the recognition system



Source: Own Elaboration

According to Figure 2 the proposal presents a context-free grammar analyzer, which aims to interpret grammars that have this type of structure. Traditionally this type of grammars begins with the symbol S and determines the beginning of the production rules, the model must identify all the non-terminal symbols that are before "=>" (produce), to analyze which are the rules to process and the terminal symbols (alphabet of the language).

The model of the lexical analyzer consists of verifying the elements that belong to the language that is represented through the terminal and non-terminal symbols, in algorithm 1 the steps that indicate the general operation for the identification of the characters are presented.

Algorithm 1. Lexical analyzer.

- Beginning
- Check if there are characters written in the input string of the grammar to be analyzed.
- If there are written characters, append in a string array the production rules unitarily, otherwise end.
- The position of the string array is incremented when a line break (/n) is found at the end of the line.
- Terminates until a line break with an empty string is encountered.
- The end

Algorithm 1 specifies four essential steps for generating the grammar, the first step of the program generates the production rules, each production rule is separated by a line break, it is represented by the symbol "/n". When a line break is located, the characters that make up the production rules are joined, the process ends when a line with 0 written characters is found. Once algorithm 1 is executed, the non-terminal symbols are located using algorithm 2.

Algorithm 2. Localization of non-terminal symbols.

- Beginning
- Read the array containing the production rules.
- Concatenate all characters before finding the "=>" symbol in each production rule.
- When the symbol "=>" is found, the previous characters are appended, and the concatenated string is saved in an array called rules.
- If the number of production items is [production count] +1 == [production count] finish, otherwise go back to the beginning.
- The end

Nonterminal symbols are usually found before the symbol "=>" this symbol is read as produces, any character found before a symbol produces, is recognized as nonterminal, and is likewise a rule of production. Algorithm 2 indicates that now of executing the analysis of the production rules previously stored by algorithm 1, all symbols after "=>" are concatenated, these symbols are stored in an array that has been called *rules*. The processing cycle ends until the end of the chain array is found.

Once the previous points have been established, the set of non-terminals will have been obtained, then the operations are executed to obtain the alphabet of the grammar, for this algorithm 3 is executed.

Algorithm 3. Obtaining the alphabet of the grammar.

- Beginning
- Read array containing production rules.
- Delete all content older than "=>", including "=>".
- All subsequent content of "=>" will be concatenated as a new production in the production chain arrangement2.
- Finish until you find the end of array elements (numelem <[count array]).
- The end

To obtain the terminal symbols, two steps are implemented, the first is contained in algorithm 3, in which the reading of the string array is executed, where all the characters after "=>" are obtained, so that the new string array has the same number of locations as the rules array. Once the new array containing the production rules without "=>" and the non-terminals prior to it is obtained, it is executed with algorithm 4.

Algorithm 4. Production rules.

- Beginning
- Replace all symbols of the production lines that correspond with the non-terminal symbols to separate the chain into *tokens*.
- Later when finishing placing the separators in the chain, generate a loop and add the tokens to the alphabet array.
- Insert until the end of each chain found in the locations of the array where the production rules of the chain are located.
- When exhausting the memory positions of the arrangement finish.
- The end

Created the alphabet array (S) of the grammar proceeds to the implementation of the string array called *wildcard*, in which the grammar is rewritten in the form of a stack structure, following the steps of algorithm 5, this new algorithm involves the implementation of a parser, which is responsible for analyzing if the strings received by the grammar present the structure described by the production rules.

Algorithm 5. Replacing elements

- Beginning
- Initialize a loop and append the elements in an *array of objects*, the non-terminal elements of the rule array are replaced by the symbol "/+*/".
- Record at each position *of the object array*, the elements that make up the production rule include all the symbols of the grammar.
- *The end*

Figure 3 Example of grammar to be stored in algorithm 5

```
S=>Trenmotor

Trenmotor=>Version_mAmodFechfErrorDDDD

Version_m=>-2.0|-1.8|-1.6|-1.2|-V2.8|-R2.8|-3.6|-3.2|-4.2|-
FsI2.0|-16V1.6|-TsI-1.8|-T1.8|-1.4|-TFsI-1.4|-TsI1.2|-1.9|-
FsI4.2|-16V1.4 Amod=>DDDD

Fechf=>DDDD
```

Source: Own Elaboration

The new array generated by algorithm 5, presents a replacement of the non-terminal elements by the symbol "/+*/", the elements containing the production rule are stored in a new location of the *object array*, in Figure 4.1 algorithm 5 is exemplified using part of the grammar. In this process the parser checks the sentences obtained that are obtained from the database. In Figure 4 some elements of Figure 3.1 are presented, which are introduced on the left side of Figure 4, the non-terminal symbols that specify the production rule become /+*/, the other symbols become equal to that of the previous arrangement and without changes in its structure.

Figure 4 Process of transformation of the original grammar

S	"=>" Nonterminal becomes	/+*/
Trenmotor	The symbol is kept	Trenmotor
Trenmotor	"=>" Nonterminal becomes	/+*/
Version_m	The symbol is kept	Version_m
Amod	The symbol is kept	Amod
Fechf	The symbol is kept	Fechf
D	The symbol is kept	D

Source: Own Elaboration

When generating the array called "wildcard", the method of evaluation of membership of a chain X is used, for this algorithm 6 is executed.

Algorithm 6. Evaluation of the chain.

- Beginning
- Read the first symbol of the "wildcard" arrangement and verify its position in the rule array, compare the position of the first element of the rule array and then verify to which production rule it belongs.
- When obtaining the rule and its position in the rules array , the lane() method is invoked to execute the path of the production options of the rule.
- If it is terminal, it is sent to be concatenated, otherwise the position of the new terminal symbol to be evaluated is returned.
- The lane() method until it finds a symbol "|" or a symbol "/+*/"
- If the string to be evaluated does not end yet go to step 2, incrementing a position in the "wildcard" array, otherwise end.
- Determine if a state of acceptance has been reached the chain under evaluation.
- The end

If the *lane() method has returned a terminal string, the* evaluate() *method verifies* the string obtained by the grammar, comparing it with the string previously entered by the application, thus obtaining the answer whether the written string belongs to the OBDII language. Figure 5 exemplifies the process of transformation of previously written grammar.

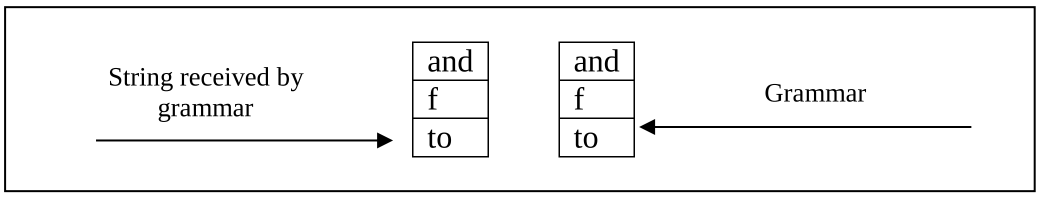
Figure 5 Grammar transformation process

Original grammar		Equivalent grammar
S	=>	/+*/
Trenmotor	----	Trenmotor
Trenmotor	=>	Trenmotor
Version_m	----	Version_
Amod	----	Amod
Fechf	----	Fechf
D	---	D
Version_m	=>	/+*/
-2.0	---	-2.0
|	----	|
-1.8	----	-1.8

Source: Own Elaboration

Figure 6 contains the string "efajk" for evaluation, on the right side of Figure 8.1 is the rule in progress that is being used to perform the language membership operation, for this example it is assumed that the production rule *S* contains the non-terminal symbols "efa". The basic principle of the stack method is to replace the characters that correspond to those in the string stack, left side of Figure 8, with those in the grammar stack, right side of Figure 8.1. At the end those characters are replaced by the symbol "#", which occupy the same position and that presents an equality in its morphology in both stacks, the above represents the characters of the string and is replaced by an empty symbol called as ø to the characters that belong to the grammar stack, the number symbols "#" and symbols "ø" must coincide between each stack.

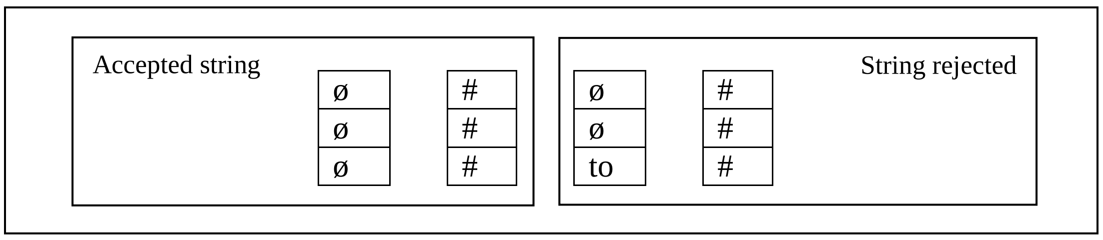
Figure 6 Implementing a Stack for String Evaluation



Source: Own Elaboration

If the entered string does not correspond to the grammar as shown in Figure 7.1, right side, while on the left side of Figure 7 a valid acceptance state has been reached:

Figure 7 Valid acceptance status of the stack left side. Invalid acceptance status of the stack, right side



Source: Own Elaboration

In Figure 8.1, it is visualized how the two stacks on the right side do not contain the same *number of elements "#" and "ø" therefore the entered string does not belong to the language, this result is obtained by initializing a counter* the number of symbols "#" exist in a stack and the number of symbols "ø" In the other, if the total number of symbols in each stack is equal to their size, a state of favorable acceptance is reached, as in the example in Figure 9.1.

Because other methods of solving grammars suggest the expansion of grammar through the syntactic tree, in this section, a similar process is exposed, but without using a syntactic tree, a mutable array is used (the number of elements can increase or decrease in real time) to achieve expansion when a non-terminal containing one or more of these is found.

On the other hand, Figure 8 shows the process of expanding one of a grammar to give rise to the generation of a response to the input string. First the character *c is stacked, then b* and at the end *a* (column I), in column II, rule *S* replaces the content to be evaluated by *aBC*, then *column III does not vary compared to column I, while the non-terminal* B *is replaced by its terminus* b (column 4). Continuing with the expansion of the grammar, the nonterminal *C*, is replaced by *c*, and remains as exemplified in column VI. The final process consists of having stacked the non-terminals are compared with the start chain, it begins to generate an equal relationship between the content of the indexes of the arrays, if they are equal, they change by the empty and # symbols. When there are 3 voids and 3 # symbols, the state of valid acceptance is reached, as it is possible to analyze in Figure 7.1.

Figure 8 Example of expanding a grammar using the stack method

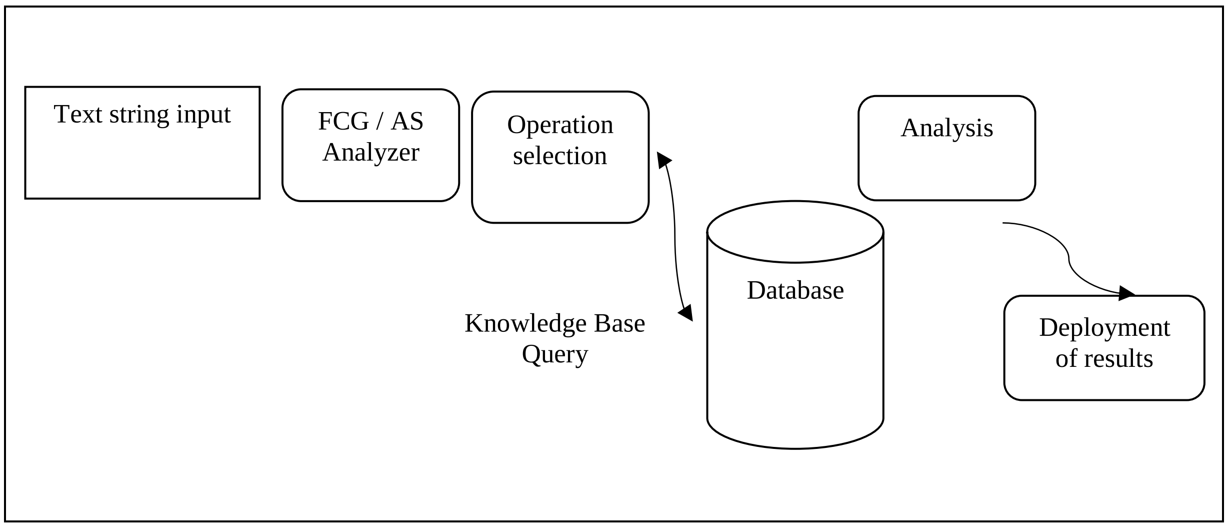
to	abc		to	to		to	to		ø	#
b		à	b	Bc		b	b		ø	#
c			c		à	c	c	à	ø	#
									3 symbols ø	3 symbols #
I	II		III	IV		V	SA W		VII	VIII

Source: Own Elaboration

Only 4 variables have been taken for the analysis of estimation of failures, because time is one of the main factors involved in the process of wear of a vehicle, as well as the operation of an engine varies from one to another, in addition the complexity of its system rises or decreases, affecting the number of OBDII codes you can present.

Figure 9 shows the general structure of the chain evaluation system, where the input is evaluated by an FCG analyzer or by sentiment analysis, from this evaluation the results are displayed by its visualization. The sentiment analyzer is based on a statistical model, which forms a structure based on a corpus of syntactic categories, to determine the classification of words, as well as to interpret based on probabilities the trend involved in the sentences that are analyzed.

Figure 9 General representation of the system with data acquisition



Source: Own Elaboration

Experimental Results and Discussion

The string recognition models for verifying language membership, and sentiment recognition have been implemented in the Python language in version 3.11, have been run on a MacBook Air M1 computer with 8 GB of RAM. For the validation of both models, the following entries have been considered.

1. Evaluate 10 text strings using context-free grammar that represents the fault code system of systems for cars that work OBDII.
2. Evaluate 10 text strings to analyze sentiment analysis.

The perspective of the analysis of the analyzer and syntactic has been raised to recognize the OBDII codes, which has an alphabet (OBDII symbols such as P, B, C, U y), remembering that all codes of this type have 5 characters. The main rule is that an OBDII code always starts with any letter either P, B, C, U, once, the other 4 symbols are a four-digit hexadecimal number, figure 10.

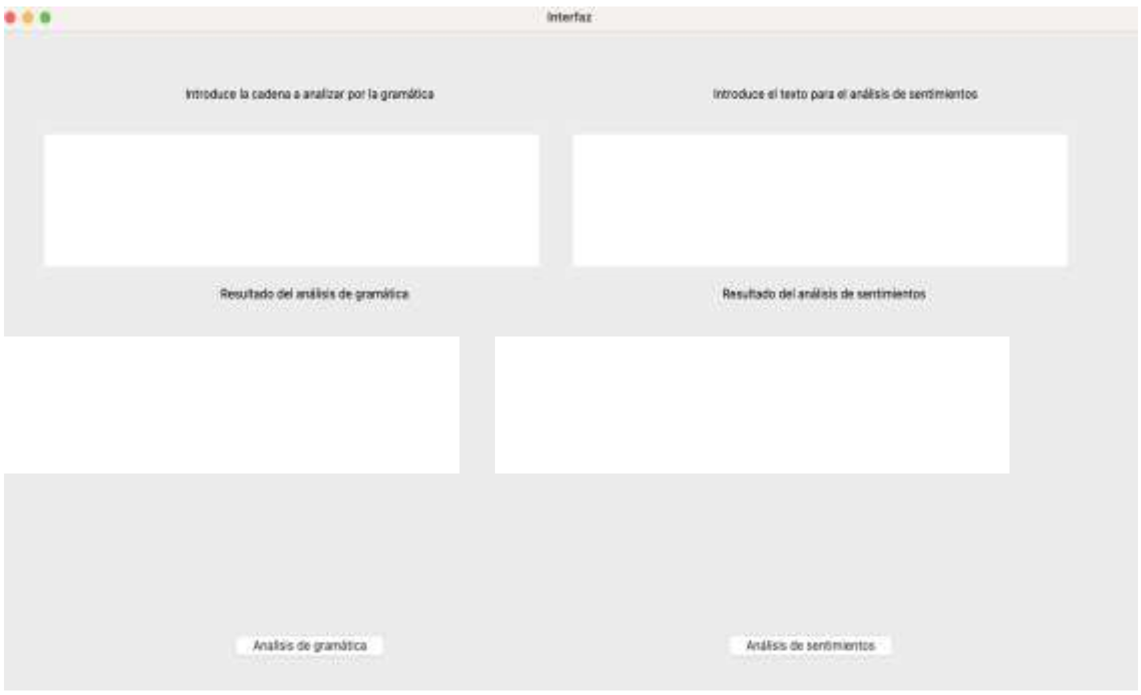
Figure 10 Grammar used for the analysis of OBDII codes.

```
S=>Trenmotor

Trenmotor=>Version_mSepAmodSepErrorDDDD

Version_m=>2.0|1.8|1.6|1.2|V2.8|R2.8|3.6|3.2|4.2-FSI2.0|16V1.6|TSI-1.8|T1.8|1.4|TFSI-
        1.4|TSI1.2|1.9|FsI4.2|16V1.4

Amod=>DDDD
```

Source: Own Elaboration

Figure 10.1 shows the interface used for the evaluation of the evaluation model. In the left section the strings to be evaluated by the grammar of Figure 11 are entered, while in the right part the section to enter the text to be evaluated by the sentiment analyzer is presented. To perform the corresponding operations, it is only required to operate the buttons that are at the bottom of the application.

Figure 11 Proposed Interface for Text Evaluation


Interfaz
Introduce la cadena a analizar por la gramática
Introduce el texto para el análisis de sentimientos
Resultado del análisis de gramática
Resultado del análisis de sentimientos
Analisis de gramática
Análisis de sentimientos

Source: Own Elaboration

Table 1 presents the evaluation results of the analyzed chains. According to the data analyzed, it was possible to observe that the results with respect to the chains corresponding to the FCG adhered to the previously established structure, resulting in the acceptance or rejection of these, on the other hand, in the analysis of feelings, the chains that tended to express a description of facts, without phrases of emotional attachment, The results tend to be classified as neutral, while sentences with words denoting rejection or acceptance will be classified as positive or negative.

Table 1 Results obtained from the analyzed chains

Grammar text evaluation	Result	Sentiment analysis text	Result
Chain 1	Accepted	Text 1	Neutral
Chain 2	Accepted	Text 2	Negative
Chain 3	Accepted	Text 3	Positive
Chain 4	Accepted	Text 4	Positive
Chain 5	Accepted	Text 5	Positive
Chain 6	Rejected	Text 6	Negative
Chain 7	Rejected	Text 7	Neutral
Chain 8	Accepted	Text 8	Neutral
Chain 9	Accepted	Text 9	Neutral
Chain 10	Accepted	Text 10	Neutral

Source: Own elaboration

The figure shows the results obtained, according to the evaluation of the text strings processed by the models, in subsection a it is shown that the sentence was analyzed as neutral in the analysis of feelings, because it expresses a definition of the mammal bears, while in subparagraphs b and c, the results indicate that the software obtained a negative and positive evaluation, respectively, with respect to the exposed sentences, finally, in subsection d of Figure 12, it is shown that the string has been accepted.

Figure 12 Results obtained from the evaluation of text strings for grammars and sentiment analysis

a) Example of sentiment analysis result with neutral result

b) Example of a positive sentiment analysis result

c) Example of sentiment analysis result with negative result

d) Example of grammar analysis result with accepted string

Source: Own Elaboration

In the tests carried out, the implemented model obtained a correct performance, because in 100% of the cases analyzed it was possible to obtain the expected recognition, as well as the appropriate classification in the analysis of feelings, although it is important to highlight that in this last part it was observed that the corpora available for the Spanish language present a lower development with respect to the corpora destined for the English language, which generates that the accuracy of the classification of phrases for the verification of the sentiment that is being expressed is lower than expected.

Conclusions

The model presented allows, on the one hand, to affirm or reject the belonging of a string to a language by reading a particular FCG, allowing it to be adaptable to different changes in grammar. On the other hand, sentiment analysis is of great importance to obtain information about the mood of people who deposit their comments in digital media such as social networks, or opinion space.

Sentiment analyzers require a large amount of data to determine if the expression read tends to be positive, negative, or neutral, in addition to that, in real applications, these require a large database to obtain relevant statistics.

Acknowledgement

We are grateful to the CONAHCYT [3873836-2022] and the Technological Institute of Higher Studies of Jocotitlán, for the support given in carrying out this research project.

References

Aschermann, C., Frassetto, T., Holz, T., Jauernig, P., Sadeghi, A.-R. & Teuchert, D. (2019). Nautilus: Fishing for deep bugs with grammars. NDSS. https://dx.doi.org/10.14722/ndss.2019.23xxx.

Brock, D. C., & Grad, B. (2022). Expert systems: Commercializing artificial intelligence. IEEE Annals of the History of Computing, 44(1), pp. 5-7. 10.1109/MAHC.2022.3149612.

Can, U. & Alatas, B. (2019). A new direction in social network analysis: Online social network analysis problems and applications. Physica A: Statistical Mechanics and its Applications, vol. 535, p. 122372. https://doi.org/10.1016/j.physa.2019.122372.

Chen, Q., Wang, X., Ye, X., Durrett, G. & Dillig, I. (2020). Multi-modal synthesis of regular expressions, in Proceedings of the 41st ACM SIGPLAN conference on programming language design and implementation, pp. 487–502. https://doi.org/10.1145/3385412.3385988.

Ching, W.-K. & Ng, M. K. (2006). Markov chains. Models, algorithms, and applications. https://doi.org/10.1007/978-1-4614-6312-2.

Chowdhary, K. & Chowdhary, K. (2020) "Natural language processing," Fundamentals of artificial intelligence, pp. 603–649. https://doi.org/10.1007/978-81-322-3972-7_19.

Denning. P. J. (1993). Machines, languages, and computation. Prentice Hall College Publishing House Div, pp. 35-56.

Dragoi, E.-N., Curteanu, S., Galaction, A.-I. & Cascaval, D. (2013). Optimization methodology based on neural networks and self-adaptive differential evolution algorithm applied to an aerobic fermentation process," Applied Soft Computing, vol. 13, no. 1, pp. 222–238. https://doi.org/10.1016/j.asoc.2012.08.004.

Galindo, E. A., Perdomo, J. A. & Figueroa-García, J. C. (2020). "Comparative study between multiclass vector support machines, artificial neu- ronal networks and self-organized neuro-diffuse inference system for classification problems," Information Technology, vol. 31, no. 1, pp. 273–286. http://dx.doi.org/10.4067/S0718-07642020000100273

Ganardi, M., Je˙z, A. & Lohrey, M. (2021). Balancing straight-line programs. Journal of the ACM (JACM), vol. 68, no. 4, pp. 1–40. https://doi.org/10.1145/3457389.

Giarratano, J. (2001) Systems Experts Principles and Programming. Third Edition. Riley-Thomson S.A. Puerto Rico, pp. 35-40.

Hemberg, E., Kelly, J. & O'Reilly, U.-M. (2019). On domain knowledge and novelty to improve program synthesis performance with grammatical evolution, in Proceedings of the genetic and evolutionary computation conference, pp. 1039–1046. https://doi.org/10.1145/3321707.3321865.

Himelboim, I., Xiao, X., Lee, D. K. L., Wang, M. Y. & Borah, P. (2020). A social networks approach to understanding vaccine conversations on twitter: Network clusters, sentiment, and certainty in HPV social networks. Health communication, vol. 35, no. 5, pp. 607–615. 10.1080/10410236.2019.1573446.

Hopcroft, J. E. Motwani, R. & Ullman, J. D. (2001). Introduction to the theory of automata, languages, and computation. First edition. Editorial CECSA, pp. 23-40.

Htet, H., Khaing, S. S. & Myint, Y. Y. (2019). Tweets sentiment analysis for healthcare on big data processing and IoT architecture using maximum entropy classifier, in Big Data Analysis and Deep Learning Applications: Proceedings of the First International Conference on Big Data Analysis and Deep Learning 1st, pp. 28–38, Springer. 10.1007/978-981-13-0869-7_4.

Huang, L., Zhang, H., Deng, D., Zhao, K., Liu, K., Hendrix, D. A. & Mathews, D. H. (2019). Linearfold: linear-time approximate RNA folding by 5'- to-3'dynamic programming and beam search. Bioinformatics, vol. 35, no. 14, pp. i295–i304. 10.1093/bioinformatics/btz375.

Kauffmann, E., Peral, J., Gil, D., Ferrández, A., Sellers, R. & Mora, H. (2020). A framework for big data analytics in commercial social networks: A case study on sentiment analysis and fake review detection for marketing decision-making. Industrial Marketing Management, vol. 90, pp. 523–537. https://doi.org/10.1016/j.indmarman.2019.08.003.

Khrais, L. T. (2020). Role of artificial intelligence in shaping consumer demand in e-commerce. Future Internet, vol. 12, no. 12, p. 226. https://doi.org/10.3390/fi12120226

Kosheleva, O. & Kreinovich, V. (2023). Why Chomsky normal form: A pedagogical note, in Decision Making Under Uncertainty and Constraints: A Why-Book, Springer, pp. 69–73. https://doi.org/10.1007/978-3-031-16415-6_10.

Murillo-Castañeda, R. A. (2021), Implementation of the vector support machines method in spatial databases for supervised classification analysis in remote sensing images. Cartographic Review, no. 102, pp. 27–42. https://doi.org/10.35424/rcarto.i102.830.

Nadkarni, P. M., Ohno-Machado, L. & Chapman, W. W. (2011). Natural language processing: an introduction. Journal of the American Medical Informatics Association, vol. 18, no. 5, pp. 544–551. https://doi.org/10.1136/amiajnl-2011-000464.

Neri, F. & Tirronen, V. (2010). Recent advances in differential evolution: a survey and experimental analysis. Artificial intelligence review, vol. 33, pp. 61–106. https://doi.org/10.1007/s10462-009-9137-2. Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. Journal of the American Medical Informatics Association, 18(5), 544-551. https://doi.org/10.1136/amiajnl-2011-000464.

Norris, J. R. (2011). Markov chains. No. 2, Cambridge university press, pp. 20-40. https://doi.org/10.1017/CBO9780511810633.

Numaya, Y., Hendrian, D., Yoshinaka, R., & Shinohara, A. (2023, July). Identification of Substitutable Context-Free Languages over Infinite Alphabets from Positive Data. In *International Conference on Grammatical Inference* (pp. 23-34). PMLR. Obtained from https://proceedings.mlr.press/v217/numaya23a.html

Qin, A. K. & Suganthan, P. N., (2005). Self-adaptive differential evolution algorithm for numerical optimization, in 2005 IEEE congress on evolutionary computation, vol. 2, pp. 1785–1791, IEEE. 10.1109/CEC.2005.1554904.

Qin, A. K., Huang, V. L. & Suganthan, P. N. (2008). Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE transactions on Evolutionary Computation, vol. 13, no. 2, pp. 398–417. 10.1109/TEVC.2008.927706

Qureshi, M. A., Asif, M., & Anwar, S. (2023). NewBee: Context-Free Grammar (CFG) of a New Programming Language for Novice Programmers. Intelligent Automation & Soft Computing, 37(1). 10.32604/iasc.2023.036102.

Russell, S. & Norving, P. (2004). Artificial Intelligence, Second Edition. ISBN:978-84-205-4003-0. Pearson Education. Madrid Spain, pp. 30-50.

Shin, E. C., Allamanis, M., Brockschmidt, M. & Polozov, A. (2019). Program synthesis and semantic parsing with learned code idioms," Advances in Neural Information Processing Systems, vol. 32. url: https://papers.nips.cc/paper_files/paper/2019/hash/cff34ad343b069ea6920464ad17d4bcf-Abstract.html
Sipser, M. (1996) "Introduction to the theory of computation," ACM Sigact News, vol. 27, no. 1, pp. 27–29.

Torr, J., Stanojevi´c, M., Steedman, M. & Cohen, S. B. (2019). Wide-coverage neural a* parsing for minimalist grammars, in Proceedings of the 57th annual meeting of the association for computational linguistics, pp. 2486–2505. https://www.aclweb.org/anthology/P19-1238.

Vayadande, K., Sangle, P., Agrawal, K., Naik, A., Mulla, A., & Khare, A. (2023, April). Conversion of Ambiguous Grammar to Unambiguous Grammar using Parse Tree. In *2023 International Conference on Inventive Computation Technologies (ICICT)* (pp. 1039-1046). IEEE. 10.1109/ICICT57646.2023.10134096

Xiao, G., Tu, G., Zheng, L., Zhou, T., Li, X., Ahmed, S. H. & Jiang, D. (2020). Multimodality sentiment analysis in social internet of things based on hierarchical attentions and CSAT-TCN with MBM network. IEEE Internet of Things Journal, vol. 8, no. 16, pp. 12748–12757. 10.1109/JIOT.2020.3015381.

Xing, H. & Qiu, D. (2009). Pumping lemma in context-free grammar theory based on complete residuated lattice-valued logic. Fuzzy Sets and Systems, vol. 160, no. 8, pp. 1141–1151. https://doi.org/10.1016/j.fss.2008.06.016.

Zucco, C., Calabrese, B., Agapito, G., Guzzi, P. H. & Cannataro, M. (2020). Sentiment analysis for mining texts and social networks data: Methods and tools. Wiley Interdisciplinary Reviews: Data Mining and Know- ledge Discovery, vol. 10, no. 1, p. e1333. https://doi.org/10.1002/widm.1333.