MAKING A GRAMMAR CHECKER WITH AUTOCORRECT OPTIONS USING NLP TOOLS

Radu Bucea Manea TONIS[6]
"Hyperion" University of Bucharest, Faculty of Economic Studies

Adrian BETERINGHE[7]
Danubius University- Galati, School of Behavioral and Applied Sciences

ABSTRACT: Our natural language approach concerns syntactic analysis using a dedicated Javascript library - wink-nlp - and semantic analysis based on Prolog programming language, facilitated by another Javascript library - tau-prolog - that allows defining logical programs, declaring rules and checking for goals inside Javascript language. Firstly, our program splits the original text into sentences, than into tokens and identifies each part of the sentence, dynamically maps entities into Prolog rules, then check the spelling accordingly to the Definite Clause Grammar (DCG) by querying the pre-defined program for initial goals (the sentence itself). Basically, we let the parser infer its own rules from the syntactic point of view, then check the grammar from a semantic perspective against the DCG inside the same work flow or pipeline of steps. The provided article combines the usage of wink-nlp and tau-prolog packages for natural language processing (NLP) and understanding (NLU).

Keywords: *grammar; natural language; logic programming; syntactic analysis*

JEL Classification: *C88; L86*

1. INTRODUCTION

Two of Chomsky's ideas were crucial for the development of our research: first, there is a basal grammar innate to the child that provides the very structures the language is built upon, and second, the recursive nature of the human language Chomsky had noticed that allows us to build an indefinite amount of statements from a finite set of grammar rules, plus the composable character of grammar that begets infinite long verbal structures. The last observation gave us the idea to establish isomorphic relations between natural language and formal systems to prove our theorems (future work).

In 1957, Noam Chomsky noticed this would be the generating principle of sentences with the famous example "Colorless green ideas sleep furiously". It follows that pairs of words have a meaning taken separately and provide grammatical structure to the sentence, even if the whole is meaningless, as shown in the following sequence (Kok & Brouwer, 2023): [["Colorless","green"],["green","ideas"],["ideas","sleep"],["sleep","furiously"]]

Grammar proofreaders fall into two categories, those that perform syntactic analysis of the sentence and ensure the identification of sentence parts in order to establish the correct

[6] PhD, Associate Prof., radub_m@yahoo.com
[7] PhD., Associate Prof. adrianbeteringhe@univ-danubius.ro

relationship between them according to a predefined linguistic model, and those that are based on AI, e.g. Grammarly, and which can learn step by step the correct structure of a sentence and transform a grammatically wrong sentence into a correct one. For learning, training sets are used, such as C4_200M made and provided by Google and which contains examples of grammatical errors along with their correct form (NLP, 2023).

Syntactic analysis shows the following aspects of the sentence (Syn, 2023):

- Word order and meaning - syntactic analysis aims to extract the dependence of words with other words in the document. If we change the order of words, then it will be difficult to understand the sentence;
- Retention of stop words - if we remove stop words, then the meaning of a sentence can be changed altogether;
- Word morphology - stemming, lemmatization will bring words to their basic form, thereby changing the grammar of the sentence;
- Parts of speech of words in a sentence - identifying the correct speech part of a word is important.

Identifying entities and their relationships in text is useful for several NLP tasks, for example creating knowledge graphs, summarizing text, answering questions, and correcting possible grammatical mistakes. For this last purpose, we need to analyze the grammatical structure of the sentence, as well as identify the relationships between individual words in a particular context. Individual words that refer to the different topics and objects in a sentence, such as names of places and people, dates of interest, or other the same, are referred to as "entities" (W1, 2023), see Figure 1:

Figure 1. The interaction between lexical analyzer and parser

Source: https://www.analyticsvidhya.com/blog/2021/06/part-11-step-by-step-guide-to-master-nlp-syntactic-analysis

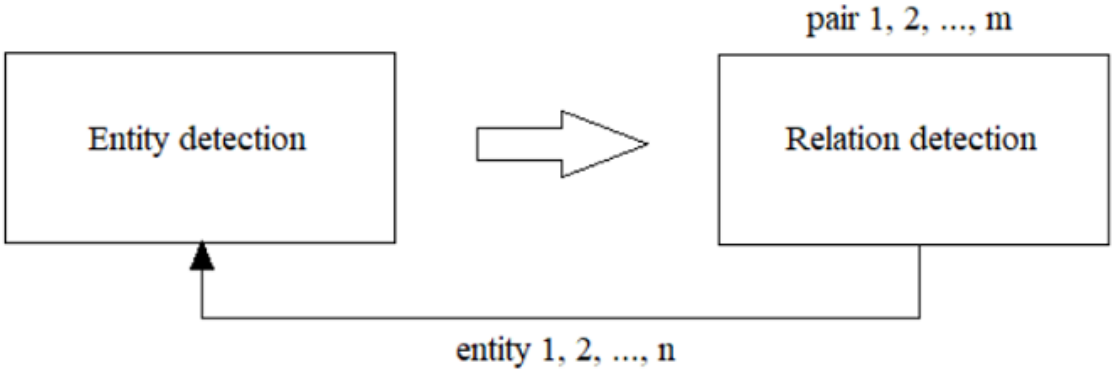
The new ES6 streaming process of transforming a text was another aspect similar to the pipeline style of the human brain in processing data. The two-way parsing on texts calculating the frequency of pairs' appearance proved to be of significant importance in dead language studies or searching for anagrams. The use of the Wink software package language model allowed us to create predicates like verbs (subject, object) based on the SVO structure of IE languages, the future knowledge base for our next proofing language system.

The agglutinative mechanism of making both sensical or not-sensical verbal content drew our attention to the Fibonacci series of numbers that is fundamental for developing living structures both at the molecular and macro level (e.g. breeding of rabbits, use of the phi constant in architecture, and so on). This way the bigrams (from n-grams) can result from concatenating strings one after another creating entities in a coherent, linked-list style, rational manner. These collocations may be interpreted either as new concepts (e.g. military-doctor) or camouflaged predicates built upon identity principles (e.g. is or exists).

Relationships are established by means of verbs or simple joining, as is the case with collocations. In the case of the latter, bigram trees can be used in the form of linear development

on the agglutinative principle or the Fibonacci sequence, resulting in simply chained lists, please see Figure 2:

Figure 2. Inference pipeline architecture



Source:https://nikhilsrihari-nik.medium.com/identifying-entities-and-their-relations-in-text-76efa8c18194

The main unit of content mapping is the sentence or statement. This way we are getting closer to a Natural Language Understanding (NLU) component responsible for extracting information at a single step throughout a pipeline process consisting of several stages (Bercaru et al, 2023): tokanization, syntactic analysis and generating the semantic grammar lexicon on the fly, based on the original term redexes, i.e. reduced forms.

2. MATERIALS

Factors such as openness, simplicity, flexibility, full browser integration, and attention to the security and privacy concerns that naturally arise in executing untrusted code have helped the Javascript language gain very significant popularity despite its low initial efficiency. Overall, it allows for a disruptive paradigm shift that gradually replaces the development of OS-dependent applications with web applications that can run in a variety of devices, some completely portable.

It should be noted that functional languages make no distinction between a fundamental parameter and a functor, moreover, most of the time (e.g. Javascript) the type is dynamically inferred. Thanks to this observation, language models have been created for the NLP based on pairs of words (bigrams), which by the order of their chaining within sentences justify a grammatical structure (Morales et al, 2012).

Javascript immutability (eg. freeze() method for objects) and local scope for variables it uses (declared with let keyword) are natural consequences of this philosophy. Generalizing the use of functors was another functional concept that merged data and functions into a single parameter, suitable for both data and behaviour interchange between objects. The arrow functions facilitated this even more by declaring a functor and initializing it in one single line of code. Another major role played here is anonymous and immediate functions (Kok & Brouwer, 2023).

Nesting functions, a feature available before ES5, were gradually replaced by currying, a special way to employ binding, a concept introduced first by the Haskel programming language. In this respect, parameters are introduced one after another inside individual parenthesis "()" suffixing the called function name, and being passed in the same order as arguments of nested anonymous functions.

All these premises made way for callback style use of Javascript that made even more room for asynchronous/event use with the advent of Promises. In this style, the nesting is achieved at the args level, explicitly providing arrow functions with anonymous

implementations instead of parameters. Implicit arguments (e.g. "a=1"), a variable number or parameters (e.g. "…args"), and memoizing, a technique enabled by closures (accessing out-of-context variables by functions) and higher order functions, which, alongside tail-calling, dramatically improve the performance of recursive functions.

Corroborating data with Popularity of Programming Language Index (PyPL) - Python, 27.7%; Java, 16.79%; Javascript, 9.65%, shows that the multi-paradigm Javascript language meets the qualities necessary for an Open source approach to natural language analysis (W2, 2023).

WinkNLP is a Javascript library for natural language processing (NLP). Specifically designed to make NLP application development easier and faster, winkNLP is optimized for the right balance between performance and accuracy. It is built from the ground up with a weak code base that has no external dependence. The .readDoc() method, when used with the default instance of winkNLP, splits text into tokens, entities, and sentences. It also determines a number of their properties. They are accessible by the .out() method based on the input parameter — its.property. Some examples of properties are value, stopWordFlag, pos, and lemma, see Table 1:

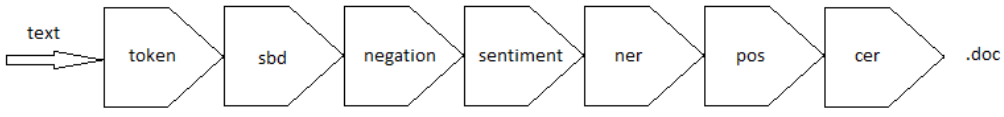
Table 1. Common its.properties that become available at each stage

Stage	Description
tokenization	Splits text into tokens.
sbd	Sentence boundary detection — determines span of each sentence in terms of start and end token indexes.
negation	Negation handling — sets the negation Flag for every token whose meaning is negated due a "not" word.
sentiment	Computes sentiment score of each sentence and the entire document.
ner	Named entity recognition — detects all named entities and also determines their type and span.
pos	Performs part-of-speech tagging.
cer	Custom entity recognition — detects all custom entities and their type and span.

Source: https://winkjs.org/wink-nlp

The readDoc() API processes input text in several stages. All steps together form a processing channel/flow, also called pipes. The first stage is tokenization, which is mandatory. Later steps such as sentence limit detection (SBD) or part-of-speech (POS) tagging are optional. Optional steps are user-configurable. The following figure and table illustrate the actual Wink flow, see Figure 3:

Figure 3. Wink processing flow



Source: https://winkjs.org/wink-nlp/processing-pipeline.html

According to (W1, 2023), there is a need for a compiler from Prolog (and extensions) to Javascript, that may use logical programming (constraint) to develop client-side web applications while complying with current industry standards. Converting code into Javascript

makes (C)LP programs executable in almost any modern computing device, with no additional software requirements from the user's point of view. The use of a very high-level language facilitates the development of complex and high-quality software.

Tau Prolog is a client-side Prolog interpreter, implemented entirely in Javascript and designed to promote the applicability and portability of Prologue text and data between multiple data processing systems. Tau Prolog has been developed for use with either Node or a seamless browser.js and allows browser event management and modification of a web's DOM using Prolog predicates, making Prolog even more powerful (W3, 2023). Tau-prolog provides an effective tool for implementing a Lexical-Functional Grammar (LFG): a sentence structure rule annotated with functional schemes such as S –> NP, VP. to be interpreted as (Frey & Reyle, 1983):

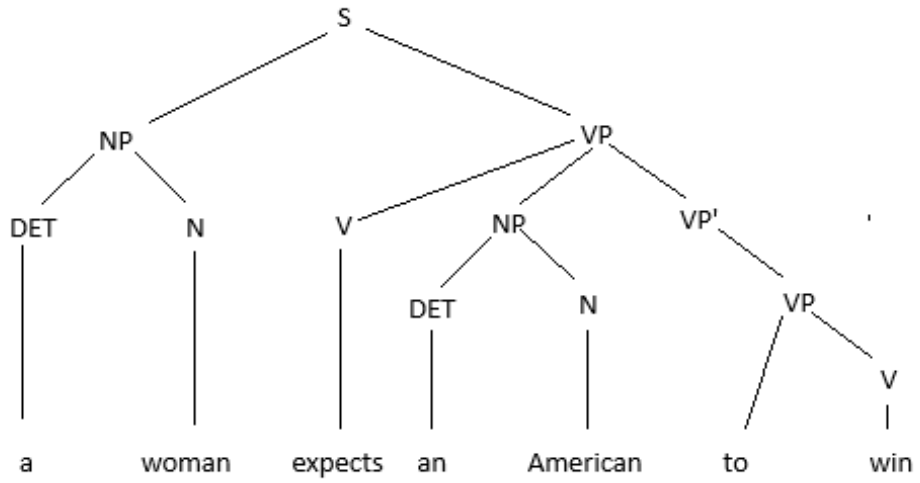
- the identification of the special grammatical relation to the subject position of any sentence analyzed by this clause vis-à-vis the NP appearing in it;
- the identification of all grammatical relations of the sentence with those of the VP.

The procedural semantics of the Prolog are such that the instantiation of variables in a clause is inherited from the instantiation given by its sub-scopes, if they succeed. Another way to deal with logic programming is using a dedicated library (W4, 2023) allowing us to declare facts and rules functional style, a step further to constraint programming, an interesting paradigm we aim to explore in our future research.

3. Methodology

After lexical analysis of the text and identification of words with the help of the token function, a first step is to identify the parts of the sentence. Extremely useful again is binary development, this time at the level of sentence, dividing the statement into noun phrase (NF) and verbal phrase (VF). Recursive development is done after the second term, decomposed into a new NF, VF and so on. For example, the process of syntactic analysis rewrites a sentence in a syntactic tree, please see Figure 4:

Figure 4. Syntactic tree



Source: Screenshot of ANTLR4 -*gui* option by author

The program loads the wink-nlp package, imports an English language model, creates a session with tau-prolog, and performs natural language processing tasks using winkNLP. It also defines a Prolog program, extracts entities from a given text, and queries the Prolog program using tau-prolog against the rules obtained by syntactic analysis (previous step).

1. The required packages and modules are imported using the require function. The wink-nlp package is imported as winkNLP, and the English language model is imported accordingly:
```
// Load required packages and modules
const winkNLP = require('wink-nlp');
const model = require('wink-eng-lite-web-model');
const pl = require("tau-prolog");
```

2. The tau-prolog package is imported as pl, and a session is created with pl.create(1000):
```
// Create a new session
const session = pl.create(1000);
```

3. The winkNLP function is invoked with the imported model to instantiate the nlp object:
```
// Instantiate winkNLP
const nlp = winkNLP(model);
```

4. The its and show variables are assigned to nlp.its and a function that logs the formatted answer from the tau-prolog session, respectively:
```
// Define helper functions
const its = nlp.its;
const showAnswer = x => console.log(session.format_answer(x));
```

5. The item variable is assigned the value of the third argument passed to the Node.js script using process.argv[2]:
```
// Get command line argument
const inputItem = process.argv[2];
```

6. The program variable is assigned a Prolog program represented as a string. It defines rules for sentence structure, including noun phrases, verb phrases, and intransitive verbs. The program also includes rules for intransitive verbs, e.g. "runs" and "laughs" (Kamath, 2015):
```
// Define the program and goal
let program = `
s(S0,S) :- np(S0,S1), vp(S1,S)
.np(S0,S) :- det(S0,S1), noun(S1,S).
vp(S0,S) :- verb(S0,S1), np(S1,S).
vp(S0,S) :- iv(S0,S).
iv(S0,S) :- S0=[walks|S].
`;
```

7. The nlp.readDoc function is used to create a document object from the inputItem. The code then iterates over each sentence and token in the document, extracting the type of entity and its part of speech:
```
const text = 'A boy eats the apples in the back. A woman runs freely on the alley';
const doc = nlp.readDoc(text);
let entityMap = new Map();
// Extract entities from the text
doc.sentences().each((sentence) => {
```

```
sentence.tokens().each((token) => {
entityMap.set(token.out(its.value), token.out(its.pos));
});
});
```

8. The extracted entities and their parts of speech are stored in a Map object as Prolog rules:

```
// Add entity rules to the program
const mapEntriesToString = (entries) => {
return Array.from(entries, ([k, v]) => `\n ${v.toLowerCase()}(S0,S) :-
S0=[${k.toLowerCase()}|S].`).join("") + "\n";
}
console.log(mapEntriesToString([...entityMap.entries()]));
```

9. The generated Prolog rules are appended to the program string:

```
program += mapEntriesToString([...entityMap.entries()]);
const goal = `
s([the, boy, eats,the,apples],[]).
`;
```

10. The session.consult function is used to load the Prolog program into the tau-prolog session. Then, the session.query function is used to query the loaded program with the specified goals. The session.answers function is used to display the answers obtained from the query:

```
session.consult(program, {
success: function() {
session.query(goal, {
success: function() {
session.answers(showAnswer);
}
})
}
});
```

4. Results

Basically, the program measures the impedance between WinkNLP and Tau-Prolog language models. It is a matter of tuning both in order to get the optimum results, this is to map and filter the output of WinkNLP according to the DCG Prolog inference rules, since the lexicon is obtained by consuming its own WinkNLP results, see the results in Figure 5:

Figure 5. The result of *corr*'s execution

```
C:\Users\radub>node corr "the boy eats the apples.the woman runs the alley."

  det([the|A],A).
  noun([boy|A],A).
  verb([eats|A],A).
  noun([apples|A],A).
  punct([.|A],A).
  noun([woman|A],A).
  verb([runs|A],A).
  noun([alley|A],A).

s([the,boy,eats,the,apples,.],[]).
s([the,woman,runs,the,alley,.],[]).
true
true
```

Source: Screenshot by author

In order to show the possible valid combination of words, it suffice changing the program's goal from 's([$sentence.tokens().out()],[]).' to 'findall(M,s(M,[]),R).'. The result will be a list of valid sentences according to the dynamic generated DCG lexicon, see Figure 6:

Figure 6. The result of *corr's findall* execution

```
C:\Users\radub>node corr "a woman runs the alley."
s([a,woman,runs,the,alley,.],[]).
R = [[a,woman,runs,a,woman],[a,woman,runs,a,woman,.],[a,woman,runs,a,alley],[a,woman,runs,a,alley,.],[a,woman,runs,the,w
oman],[a,woman,runs,the,woman,.],[a,woman,runs,the,alley],[a,woman,runs,the,alley,.],[a,alley,runs,a,woman],[a,alley,run
s,a,woman,.],[a,alley,runs,a,alley],[a,alley,runs,a,alley,.],[a,alley,runs,the,woman],[a,alley,runs,the,woman,.],[a,alle
y,runs,the,alley],[a,alley,runs,the,alley,.],[the,woman,runs,a,woman],[the,woman,runs,a,woman,.],[the,woman,runs,a,alley
],[the,woman,runs,a,alley,.],[the,woman,runs,the,woman],[the,woman,runs,the,woman,.],[the,woman,runs,the,alley],[the,woma
n,runs,the,alley,.],[the,alley,runs,a,woman],[the,alley,runs,a,woman,.],[the,alley,runs,a,alley],[the,alley,runs,a,alle
y,.],[the,alley,runs,the,woman],[the,alley,runs,the,woman,.],[the,alley,runs,the,alley],[the,alley,runs,the,alley,.]]
false
```

Source: Screenshot by author

It is important to notice that a determinant like 'a', i.e. ∃, almost triples the area of semantic field, thus emphasizes the importance of the semantic capabilities of the parser. It is obvious we have to run the findall method after each sentence not to combine the lexicon of the two sentences. Otherwise, the result is interesting, bring our program closer to generating AI features, e.g. chatGPT, rather than a normal grammatical corrector: the boy eats the boy , the boy eats the apples , the boy eats the woman , the boy eats the alley , the boy runs the boy , the boy runs the apples , the boy runs the woman , the woman eats the apples , and so on. This is most likely the field of AI (e.g. https://sunilchomal.github.io/GECwBERT/#c-bert) to choose the appropriate language model in order to get the minimum entropy or information loss.

CONCLUSION

It follows from various studies, including our own, that use of pure functional languages ensures the safe use of programs by guaranteeing there are no state changes in software execution (Khanfor & Yang, 2017). If the required packages (wink-nlp, wink-eng-lite-web-

model and tau-prolog) are not installed, the code will throw an error. Also, if the Node.js script is not executed with a third argument, the item variable will be undefined, which may cause issues later in the code. Our approach aimed to make this class of programs as accessible and useful as possible to achieve the intended purpose. It is a fact that Javascript multi-paradigm language has provided us with a more concise, more familiar, and easier language to program in, allowing the writing of NLP algorithms in a style much closer to that of conventional functional programming languages. In our future research will add error handling to gracefully handle any exceptions thrown during package imports or function invocations, and, eventually, implement additional natural language processing tasks using the wink-nlp package.

REFERENCES

1. Bercaru, G., Truică, C.-O., Chiru, C.-G. and Rebedea, T. (2023), "Improving Intent Classification Using Unlabeled Data from Large Corpora". Mathematics 2023, 11, 769. https://doi.org/10.3390/math11030769.
2. de Kok & D., Brouwer, H. (2023), "Natural Language Processing for the Working Programmer: https://www.researchgate.net/publication/2
3. Frey W. & Reyle U. (1983), "A Prolog Implementation of Lexical Functional Grammar as a Base for a Natural Language Processing System". Conference of the European Chapter of the Association for Computational Linguistics (1983); URL: https://api.semanticscholar.org/CorpusID:17161699
4. Kamath, R, Jamsandekar, S. and Kamat, R. (2015), "Exploiting Prolog and Natural Language Processing for Simple English Grammar". In Proceedings of National Seminar NSRTIT-2015, CSIBER, Kolhapur, Date of Conference (March 2015); URL: https://www.researchgate.net/
5. Khanfor, A. & Yang, Y. (2017), "An Overview of Practical Impacts of Functional Programming", 24th Asia-Pacific Software Engineering Conference Workshops, DOI: 10.1109/APSECW.2017.27, URL: https://www.researchgate.net
6. Morales, J.F., Haemmerlé, R., Carro, M. and Hermenegildo. M.V. (2012), "Lightweight compilation of (C)LP to JavaScript", Theory and Practice of Logic Programming 2012, 12(4-5), 755–773, https://doi.org/10.1017/S1471068412000336. Krill, P. (2023).
7. NLP (2023), "Building a Grammatical Error Correction Model". https://towardsdatascience.com/nlp-building-a-grammatical-error-correction-model-deep-learning-analytics-c914c3a8331b
8. Syntactic Analysis (2023), "Guide to Master Natural Language Processing (Part 11)". https://www.analyticsvidhya.com/blog/2021/06/part-11-step-by-step-guide-to-master-nlp-syntactic-analysis
9. W1 (2023), "Relation Extraction and Entity Extraction in Text using NLP". https://nikhilsrihari-nik.medium.com/identifying-entities-and-their-relations-in-text-76efa8c18194
10. W2 (2023), "C++ still shining in language popularity index", InfoWorld, https://www.infoworld.com/article/3687174/c-still-shining-in-language-popularity-index.html
11. W3 (2023), "An open source Prolog interpreter in JavaScript". https://socket.dev/npm/package/tau-prolog
12. W4 (2023), "Logic programming in JavaScript using LogicJS". https://abdelrahman.sh/2022/05/logic-programming-in-javascript