

TELECOM
Paris



 **IP PARIS**

July 2, 2023

SD-213
Cognitive Approach
to Natural Language Processing

Micro-studies

teaching.dessalles.fr/CANLP

This document contains students' contributions written during the course *Cognitive Approach to Natural Language Processing* taught in April-June 2023. I am very thankful to students for having be active participants, and also for the quality of their work

Jean-Louis Dessalles

Contents

Fatima	Balde	Building the phylogeny of the current genera of felines	3
Augustin	Curinier	Extending the procedural semantics lab work on chess.	7
Victor	Darré	Building a database of Mario World	11
Axalia	Levenchaud	Building a database of Mario Wolrd	11
Andrey	El-bitar	Russian grammar and syntax	17
Alex	Elenter	Building family tree from Text	21
Pauline	Escavi	Italian Grammar	25
Thang	Bui Doan	Vietnamese parser	29
Roel	George	Vietnamese parser	29
Olivier	Jasson	Recognizing punctuation	33
Antoine	Andurao	Procedural Semantincs in Game of Thrones	37
Lisa	Lloret	Procedural Semantincs in Game of Thrones	37
Romuald Albert	Noubissi	Building grammar for football player's position	43
Léon	Sillano	Building grammar for football player's position	43

Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Name: Fatima Balde

Phylogeny of the current genera of felines

June 25, 2023

Abstract

Abstract

For the SD213's project I choose to extend the lab of Procedural Semantic by creating a knowledge base about the phylogeny of the current genera of felines. The goal of my project is to recognize sentences that talk about the phylogeny of felines such as "the lion is a panthera" and also to detect false affirmations such as "the panthera is a lion". For an extension, I will also answer questions such as 'What are lions' by giving the current parent of lions in the phylogeny tree.

Problem

The basic idea was to create an interactive game that would teach feline phylogeny to users. The two main problems I faced were :

- The problem of structure: the sentences proposed by the user must be well structured. The words in the sentence must be well arranged. You can't say 'the is lion a felina' etc...
- The problem of semantics: sentences must have a correct meaning. For example, the lion is a panthera but the panthera is not a lion. Also, negation sentences must be taken into account the cat is not a panthera must be true.

And finally, we had to add a system to answer the user's questions. Typically, if the user asks what a lion is, the answer should be felinae.

Method

I used the logic of the procedural semantics tp that I developed to meet the needs of my project.

First I set up my own grammar. There are all the animals in the feline lineage, the verb 'to be' in the singular and plural, and the negation mark not.

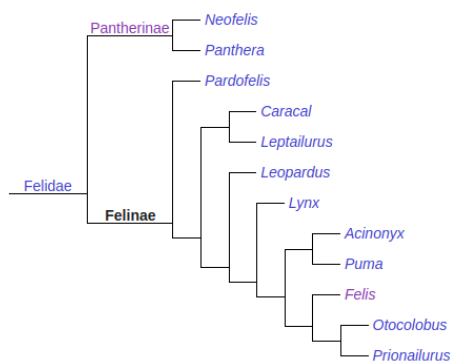
Then I defined all the rules for recognising that a sentence is correct. Since my sentences don't have the same structure as those in the tp, I redefined my own rules to suit my project. Then I defined a word file in which I put all my predicates and rules allowing me to make the linkage and also to answer the user's questions.

Finally, I took the semmain file (That I changed a little) to run my predicates

Results

- Sentences which don't have a good structure are well detected
- I do recognize true sentences .
- False sentences are well recognized and well detected
- Negation is well handled
- plural sentences are well handled
- I get response when I ask questions

Phylogénie des genres actuels de félins d'après Johnson et al. (2006)¹²:



```

linking be( _11046, _11048, _11050) with felinae( _11132)
...> felinae(felinae)
linking be( _11046, felinae, _11050) with lion( _10940)
...> lion(lion)

Syntactically correct
[gloss:be,num:sing,pers:3,subj:dp([gloss:lion,num:sing]),cpl:[adv(not)]]

s
|_ dp
|   |_ det : the
|   |_ np
|       |_ n : lion
|   |_ vp
|       |_ v : be
|       |_ dp
|           |_ det : a
|           |_ np
|               |_ n : felinae

false.
  
```

Figure 1: Feline phylogeny , recognizing true sentence(the lion is a felinae is false)

```

Sentence > the lion is not a felinae
s( 8858, 8864, 8870,[the,lion,is,not,a,felinae],[])
linking be( 9080, 9082, 9084) with not
linking be( 9080,not, 9084) with felinae( 9200)
...> felinae(felinae)
linking be( 9080,not,felinae) with lion( 8974)
...> lion(lion)

Syntactically correct
[gloss:be,num:sing,pers:3,subj:dp([gloss:lion,num:sing]),cpl:[adv(not)]]

s
├── dp
│   ├── det : the
│   └── np
│       └── n : lion
└── vp
    ├── v : be
    ├── adv : not
    └── dp
        ├── det : a
        └── np
            └── n : felinae

...> be(lion,not,felinae)
--> be(lion,not,felinae)
this sentence makes sense
type [:] to see alternatives...

```

```

Sentence > lions are pantheras
s( 9818, 9824, 9830,[lions,are,pantheras],[])
linking be( 10002, 10004) with panthera( 10052)
...> panthera(panthera)
linking be( 10002,panthera) with lion( 9900)
...> lion(lion)

Syntactically correct
[gloss:be,num:plur,pers:3,subj:dp([gloss:lion,num:plur]),cpl:[np( 9998)]]

s
├── dp
│   └── np
│       └── n : lion
└── vp
    ├── v : be
    └── dp
        ├── np
        └── n : panthera

...> be(lion,panthera)
--> be(lion,panthera)
this sentence makes sense
type [:] to see alternatives...

```

Figure 2: Handling negation , recognizing true sentence

```

Sentence > lions are not felinaes
s( 11478, 11484, 11490,[lions,are,not,felinaes],[])
linking be( 11662, 11664) with felinae( 11740)
...> felinae(felinae)
linking be( 11662,felinae) with lion( 11560)
...> lion(lion)

Syntactically correct
[subj:dp([gloss:lion,num:plur]),num:plur| 11818]

s
├── dp
│   └── np
│       └── n : lion
└── vp
    ├── v : be
    ├── adv : not
    └── np
        └── n : felinae

linking be( 11662, 11664, 11666) with felinae( 11742)
...> felinae(felinae)
linking be( 11662,felinae, 11666) with lion( 11560)
...> lion(lion)

```

```

Syntactically correct
[gloss:be,num:plur,pers:3,subj:dp([gloss:what]),cpl:[adv(not)]]

s
├── qad : what
└── vp
    ├── v : be
    └── dp
        ├── np
        └── n : lion

panthera
...> be(what,lion,panthera)
--> be(what,lion,panthera)

```



Figure 3: Plural sentence(Lions are not felinaes) , You get response when you ask questions(what are lions , I get response panthera)

Discussion

I did get all the response that I wanted and the implementation did go well. To go forward , I tried to implement cases where you recognize sentences like "lions and tigers are felinaes" but I didn't have time to implement it.

Bibliography

Tp Procedural Semantics

 	<p style="text-align: right;">June 2023</p> <p style="text-align: center;">SD-213 Cognitive Approach to Natural Language Processing Micro-study</p> <p style="text-align: right;">teaching.dessalles.fr/CANLP</p>
--	--

Name: Augustin CURINIER

Chess semantics

Abstract

I would like to add some Prolog code in order to create a symbolic "AI" that could understand more sentences about chess, such as moves and openings, and see if they make sense or not in the context of the game. This is an extension of the lab about procedural semantics that tackled the subject of chess.

Problem

The program should be better at understanding sentences and phrases related to chess, not fail in case of ambiguity and check if a move is legal or not. For instance in the lab, we had a problem with the dp “the pawn to the right of the black pawn” that rendered inaccurate results. We would like to avoid this, for example by reimplementing the execute/1 predicate in the “sem_main.pl” in order to view right(Loc) as a function rather than a mere predicate. Then, we need the program to understand more advanced concepts about chess that imply movements and not only assessing a static position.

Problem is, we have to teach the rules of chess to the computer in order to avoid illegal moves, but also ensure that what we ask to the computer makes sense in terms of NLP.

Method

In order to do so, I basically enhanced the glossary in the 'sem_grammar.pl' file to increase the knowledge related to the game of chess: the concepts of move and capture, and later the ones of check and checkmate.

First thing was to add 2 relations, other than left and right: above and below, as there are 4 possible directions in a chess board. However, those are prepositions and not nouns, which requires different processing than the first directions.

This presents several challenges. First of all, each piece moves differently, which will require a specific implementation for each one of them. To begin with, we update the glossary with the verb 'to move' which will be the same no matter the piece.

Then, we need to add the other side of the 'move' concept which will be the predicate move/2 that will take 2 locations as arguments: the starting location and the end location. This predicate will need to check at first whether or not this move is possible, and it can be quite complicated because it depends: on the position of the piece that would be moved, the positions of other pieces around it, and on the move capacity of the piece itself!

Also, a piece can capture another one or realize a check.

Results

It has been quite difficult to obtain satisfying results. I have tried different implementations, but none of them worked really well. One idea would maybe to store the pieces and their locations in a list named 'Board' in order to facilitate the process.

The problem I encountered mainly was that it was not merely about coding a prolog chess engine, it had to be able to process semantic linking also, and therefore modify the grammar accordingly.

For example, when it was about to simply add 2 new directions, there was a semantic problem as above and below are not used in the same way as left and right in the English language.

```
up((HP, VP1), (HP, VP0)) :-                               % VP1 is above HP0
    nonvar(VP0),
    VP1 is VP0 +1 .
up(Loc, Piece0) :-    % metonymic use: replacing Piece0 by its location
    nonvar(Piece0),
    Piece0 =.. [_, _, Loc0],
    up(Loc, Loc0).

down((HP, VP1), (HP, VP0)) :-                             % VP1 is below HP0
    nonvar(VP0),
    VP1 is VP0 -1 .
down(Loc, Piece0) :-    % metonymic use: replacing Piece0 by its location
    nonvar(Piece0),
    Piece0 =.. [_, _, Loc0],
    down(Loc, Loc0).
```


Therefore, the program was always rendering ‘False’ when asking ‘above the white knight’ or something similar.

It was needed to change the grammar rules in order to allow semantic linking for prepositions, but would have caused a problem as some of them like ‘to’ do not carry any predicate information.

Having already a problem there, it was difficult to obtain something for the movements, however I have written a bit of code and ideas on how to implement them if it had been possible.

We would in fact need a general ‘move’ predicate that would take locations but also the type of piece moved as an argument, and then redirect to the adequate one → ‘move_pawn’, ‘move_queen’, ‘move_knight’ ...”

I have also tried to solve the predicate vs function error on “the pawn to the right of the black pawn” sentence by modifying the ‘execute predicate’ but it raised some ‘callable’ errors.

Discussion

I expected this to be easier to implement, as there exist already chess implementations in Prolog. However, the semantic linking with natural language rendered the task way harder than I thought, adding necessity to think about the formulation of sentences in order to ask the computer for an action.

I think that in order to work it might be needed to change some grammar rules, in order for the program to be more flexible and do more things.

Bibliography

<https://boxbase.org/entries/2018/nov/19/modeling-chess-in-prolog/>

<https://github.com/he7850/Prolog-chess-game>

https://wiki.ubc.ca/Chess_Game_Prolog

https://aicourse.r2.enst.fr/CANLP/tp_ProceduralSem.html



SD213

Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Names: Axalia LEVENCHAUD and Victor DARRE

Building a mini knowledge base of New Super Mario World

June 29, 2023



Abstract

This project was carried out as part of a mini-study in the class of Cognitive Approach to Natural Language Processing (SD213). The goal of our project is to compute a mini-knowledge base on the game “New Super Mario Bros”, inspired by the lab work on Procedural semantics.

Problem

"New Super Mario Bros" is a popular platformer video game released back in 2006 for the Nintendo DS handheld gaming system. The game features 8 amazing worlds, each with their own unique theme. The goal of the game is to save the princess Peach, who has been kidnapped by the main character Mario's ultimate enemy : Bowser.

Here, the main goal of our project is to create a mini-knowledge base that is going to :

- identify each character of the game based on physical characteristics, by recognizing sentences such as :

"the character with a red hat."

- identify actions made between two character :

"the pink princess likes the red character."

- identify a character based on his relationship with an other :

"the green lizard fights the enemy of the red character."

Method

The database

To implement the "New Super Mario Bros" universe in our program, our group created a file "Character.pl" that lists all of the characters of the game thanks to the information that we could have on the Internet. We have 18 characters in total. Each character was listed in a category "character". In addition, we added many specialization on the character such as : the gender of the character the species of the character physical traits, such as the clothes the character wear and the color of the clothes the size of the character the relationship that each character have with the main character Mario Those informations were type by hand, to add the physical characteristics of each character that were not mentioned in the official website.

Addition to the existing code

Our group added specifications in the lexicon in the file "sem_Grammar.pl" to match the information that we added.

Results

For the beginning, we wanted to check if the program did recognize each character based on the additional information that we gave into the file "Character.pl". When running the code and typing the sentence:

the character with a red hat.

We got :

```
Syntactically correct
[gloss:character,num:sing]

dp
|__det : the
|__np
|__n : character
|__pp
|__p : with
|__dp
|__det : a
|__np
|__adj : red
|__np
|__n : hat

----> character(Mario)
this sentence makes sense
type [;] to see alternatives... █
```

We also wanted to check if exemple from the past would still work with our new grammar. As we run the code and type the sentence :

the pink princess likes the red character

We got :

```
Syntactically correct
[gloss:like,num:sing,pers:3,subj:dp([gloss:princess,num:sing]),cpl:[dp([gloss:character,num:sing])]

s
|__dp
|__det : the
|__np
|__adj : pink
|__np
|__n : princess
|__vp
|__v : like
|__dp
|__det : the
|__np
|__adj : red
|__np
|__n : character

----> like(Peach,Mario)
this sentence makes sense
type [;] to see alternatives... █
```

We added information about the relationship that exists between the main character Mario and each character. As we run the code and type the sentence :

the brother of the red character helps the pink princess

We got :

```
Syntactically correct
[gloss:help,num:sing,pers:3,subj:dp([gloss:brother,num:sing]),cpl:[dp([gloss:princess,num:sing])]]

s
|__dp
|  |__det : the
|  |__np
|  |  |__n : brother
|  |  |__pp
|  |  |  |__p : of
|  |  |  |__dp
|  |  |  |  |__det : the
|  |  |  |  |__np
|  |  |  |  |  |__adj : red
|  |  |  |  |  |__np
|  |  |  |  |  |  |__n : character
|  |__vp
|  |  |__v : help
|  |  |__dp
|  |  |  |__det : the
|  |  |  |__np
|  |  |  |  |__adj : pink
|  |  |  |  |__np
|  |  |  |  |  |__n : princess

---> help(Luigi,Peach)
this sentence makes sense
type [;] to see alternatives... █
```

Also by adding new intercatation verbs, we could have result like with the sentence :

the green lizard fights the enemy of the red character

That gives :

```
Syntactically correct
[gloss:fight,num:sing,pers:3,subj:dp([gloss:lizard,num:sing]),cpl:[dp([gloss:enemy,num:sing])]]

s
|__dp
|  |__det : the
|  |__np
|  |  |__adj : green
|  |  |__np
|  |  |  |__n : lizard
|  |__vp
|  |  |__v : fight
|  |  |__dp
|  |  |  |__det : the
|  |  |  |__np
|  |  |  |  |__n : enemy
|  |  |  |  |__pp
|  |  |  |  |  |__p : of
|  |  |  |  |  |__dp
|  |  |  |  |  |  |__det : the
|  |  |  |  |  |  |__np
|  |  |  |  |  |  |  |__adj : red
|  |  |  |  |  |  |  |__np
|  |  |  |  |  |  |  |  |__n : character

---> fight(Yoshi,Bowser)
this sentence makes sense
type [;] to see alternatives... █
```

Discussion

Interesting aspects of our work that could have been developed are :

- Expanding even more the database so it contains more information on character. One of the main work would be to add the relationship between characters that are not the main character Mario. Also adding other types of interaction with other characters : jump on a Koopa and it kills him, but also drop a shell.
- Regarding the work done on other labs, we could have extended our project to the other aspects of Natural Language Processing such as Processing Aspect, relevance and argumentation to create dialog about the New Super Mario Bros game. For example, we could use the CAN (Conflict-Abduction-Negation) procedure to create dialog about how to beat the game.

Example : [Context : A wants to finish the game and asks B to know what to do with the character in order to]

A : I would like to finish the world I am in. I need to finish every world to defeat Bowser because the goal of the game is to save the princess Peach.

B : You have to go to the castle at the end of the world you are in.

A : Yes but I don't know where the castle is.

B : All the castles are on the right, so your character needs to move to the right of your screen.

A : But there are enemies on the way to the castle.

B : You need to kill them.

A : But I don't know how to kill them.

B : You can jump on them to kill them without damage.

- Question answering, which could have been a complementary task to the two aspects developed above. In our initial plan, we wanted to implement this aspect to obtain answer on how to win the game, but we didn't manage to put it into practice. By extending the grammar rules and creating an appropriate answer to a question, this work could have added a more fun aspect to our mini-project.

Example :

“*How to win the game*”, obtain “Mario should saves the princess Peach”

“*How to save the princess Peach*”, obtain “Mario should kill Bowser”

“*How to kill Bowser*”, obtain “Mario should finish all the world”

Limits

We encountered difficulties when running the code and typing a sentence with words that were common to multiple character such as :

the character with a red hat

Not only we got the character Mario, but also the character Baby Mario. To patch this problem, we wanted to add size characteristics such as tall and small. But the program doesn't understand when two adjectives are put together.

Also, when typing :

the yellow princess

the program identifies Daisy and Wario, who has not been identified as a princess.

An other limit to our code is about the reversibility of the relationship that we created. In fact, when seeing other alternatives to our sentence parsing, relationships are switching their position :


brother(Luigi, Mario) became brother(Mario, Luigi)

In this case, the reversibility is not relevant, but considering relationship such as "girlfriend", the reversibility of the relationship become inadequate, knowing that Mario cannot be the girlfriend of the princess Peach.

Conclusion

This mini-project was a good opportunity to look into the details of the lab on Procedural semantics, and making our own project version with a fun subject. It was also an opportunity to be confronted with other difficulties that we could have with procedural semantics.

Even if all the initial objectives were not accomplished, we were satisfied to correctly implement our database in the previous lab work and successfully infer the meaning of the subjects and predicates in a sentence.

  	<p>May 2023</p> <p>SD-213</p> <p>Cognitive Approach to Natural Language Processing</p> <p>Mini-project</p> <p>teaching.dessalles.fr/CANLP</p>
---	--

Name: Andrey EL-BITAR

Russian semantic grammar

Abstract

This mini-project tends to implement the grammar structure of a different language from those seen during the course. As a native Russian-speaker, I'm not completely aware of the grammatical and syntax structure, that is an intuition for me but a major barrier for my friends who learn Russian. It is interesting to concretize it and analyze the behavior of a non-latin language.

Problem

The Russian language seems to be hard to learn because of its flexible word order structure and complex 6-cases grammar. Moreover there are no determiners and their function is completed by a lot of noun ending declinations. Implementing this grammar will show how those changes impact the performance of approaches seen in SD213. Hopefully, Prolog is able to carry cyrillic alphabet, so we can focus on the core challenge.

Method

To begin with, I tried to extend the first lab (simple grammar) structure. As expected it was almost useless because of the Russian ever-present ending declinations, that depend on the semantic structure of the sentence. Thus, I focused on

the semantic grammar code, changing it to manage main differences between languages:

- Besides the singular/plural contract, we should add three genders : masculine, feminine and neutral (like in German). By the way, the pronoun's gender is defined by the subject (like in English) and not the object (like in French).
- Unlike the most European languages, the cases in Russian are always used, changing the word ending, so that we have to add all six and check them every time. An automated code generator program will be needed if we want to extend the available vocabulary space and approach some declination rules.
- The 'be' verb doesn't appear in the present, which sensitively reduces the basic sentence pattern.
- Although there are no determiners in Russian, their function can be held by possessive and demonstrative pronouns.

Results

The script already recognizes (at all genders and numbers), sentences having the structure-shape similar to:

- 'Дима большой' = 'Dima is big' (proper noun + [is] + adj)
- 'она красивая' = 'she is beautiful' (personal pronoun + [is] + adj)
- 'соседи весёлые' = 'the neighbors are funny' (common noun + [is] + adj)
- 'его соседи весёлые' = 'his neighbors are funny' (possessive pronoun + [is] + adj)
- 'эти соседи весёлые' = 'those neighbors are funny' (demonstrative pronoun + [is] + adj)
- 'мой Томер будет красивым' = 'my Tomer will be beautiful' (linking verb that requires a complement at the instrumental case)

```
Sentence > мой брат думает о ней
Syntactically correct
[gloss:думает,num:sing,pers:3]

s
|--dp
|  |--det
|  |  |--posp : мой
|  |  |--dp
|  |     |--comp : брат
|  |--vp
|     |--triv : думает
|     |--dp
|        |--p : о
|        |--dp
|           |--prep : ней
_6484
this sentence makes sense
```

```
Sentence > моя соседка любит тебя
Syntactically correct
[gloss:любит,num:sing,pers:3]

s
|--dp
|  |--det
|  |  |--posp : моя
|  |  |--dp
|  |     |--comp : соседка
|  |--vp
|     |--trdv : любит
|     |--dp
|        |--prep : тебя
_4126
this sentence makes sense
```

- 'моя соседка любит тебя' = 'my (female) neighbor loves you (direct transitive verb that requires a complement at accusative case)

- 'мой брат думает о ней' = 'my brother thinks about her' (indirect transitive verb that requires a prepositions followed by a complement at an instrumental case.)

Discussion

Although the main structural semantic tasks are well completed by my script, a few paths of improvement can be listed.

First, I faced some problems when trying to combine several structure tasks. I think there may be a conflict of clauses, and their order should be rethought. Generalizing the grammatical structure may help a lot in reducing the number of clauses, making the script more compact and simple. But this refers to a more profound linguistic work of language mechanics.

Second, when encoding some vocabulary, I found it very tough to express new words, which seems to follow the same rule. It might be clever to encode a data generator script that will use an already existing database, and automatically manage word endings depending on genders and cases. The real stuff will be to find a trade-off between rule exceptions and rule generality, maybe ignoring some words at first attempt.

Here, I focused on the implementation of gender and case semantic interactions between terms. Basically, I didn't find out how to manage the flexible word order using symbolic NLP, which is an interesting question to dig on. A possible way of exploring this is to rethink the semantic linkage between terms, so that the plotted function tree can counter the order-less structure of Russian sentences.

Bibliography

- [wikipedia](#)
- [6 cases in russian](#)
- [verb types](#)
- [grammatical properties and lexical database](#)



SD213

Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Building the Simpsons Family Tree from Text

Alex EENTER

June 15, 2023

Abstract

We managed to build a system that was able to take a raw text as input and build a Prolog relational database representing a family tree. The system is based on procedural semantics.

Problem

To put in simple words we are trying to build a system that *understands* a text which describes a family tree. The word *understand* is a complicated word to define, however as the reader will notice the system manages to build a reasonable relational database using little context. The main idea of the work is to show the power of procedural semantics. Moreover, a nice side goal is to show the power of the Prolog language. With fairly simple code a seemingly complicated system is built.

As Gregory Chaitin puts it: "What's the smallest self-delimiting program that produces a given output? The size of that program in bits is the complexity $H(\text{output})$ of the output". Using Prolog we get a fairly short program that produces the desired output, therefore something that is complex in hindsight (a family tree) becomes algorithmically not so complex. This shows us the power of procedural semantics and Prolog.

Method

To build the described system an analog approach to the semantics lab performed during the course was used. The following subsections describe how the system works.

Background Knowledge

The system is first provided with context. The context are concepts (words) which will be fed to the system as sentences. These concepts may be seen at the end of the *sem_family.pl* file. Each concept presents feature structures. These features are key to our work. The features describe firstly which words can be *combined* in the same sentence. This allows us to parse only well-conjugated phrases. Moreover, we will use a feature to determine how semantic linking should be done. This will be explained in a following section.

DCG Parser

The system may be seen as a big DCG Parser. The interesting part of the work is not in the DCG clauses themselves but in the Prolog code that is embedded. As syntactic parsing was not the objective of our work we will not further study these side. However, we may note that Prolog allows us to build DCG clauses in a fairly simple and elegant way.

Procedural Semantics

Procedural semantics may be seen as linking a truth value to a defined relationship. In other words, if the system successfully parses `child(Bart,Homer)`. We understand it is true that Bart is the child of Homer. The method used in the work is semantic linking. Every familiar relationship is represented as *relationship(Participant1,Participant2)*. As the example described earlier.

To exemplify let's describe how the phrase "Bart is the child of Homer" is analyzed. The tree is shown in Figure 1. Looking at the features of the verb *be* we find: *be(,_,_)*. This means that there are 3 semantic links to be done. At different nodes in the tree the linking will be done, arriving at the final composition of *be(child, Bart, Homer)*. Which will then be written in the database as *child(Bart, Homer)*. Seeing the tree bottom-up, the first link is done in *vp → v, dp, pp* which results in *be(child, _, Homer)*. The second when with *s → dp, vp* ; which gives the desired result.

The next step in our work was to *understand* many to one relationships. Let's again analyze an example: "Patty and Selma are the sisters of Marge". The parsed tree is shown in Figure 2. The procedure is the same as in the last example, however when performing

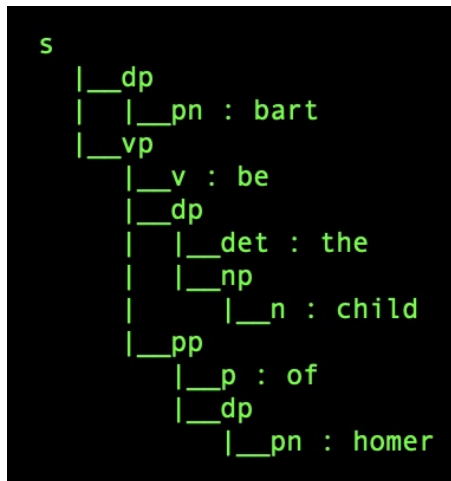


Figure 1: Parse Tree: "Bart is the child of Homer"

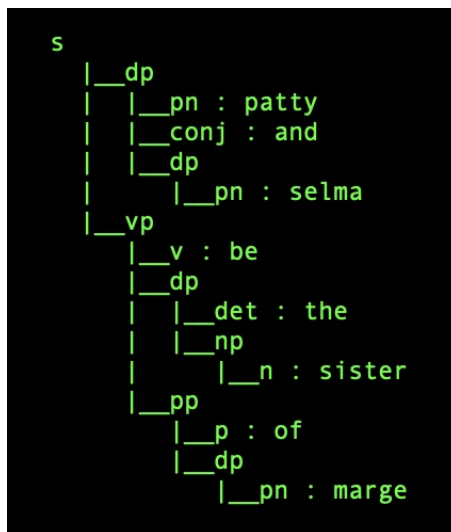


Figure 2: Parse Tree: "Patty and Selma are the sisters of Marge"

the linking at step $s \rightarrow dp, vp$; we will end up with the second argument being a list $be(sister, [Selma, Patty], Marge)$. Later on, we will introduce two facts to our database $sister(Selma, Marge)$ and $sister(Patty, Marge)$.

Parsing a Text

To parse a whole text we make use of Prolog's fail loops. We basically force the parser to fail after parsing each sentence, in order to parse the next one. The same idea is used to insert multiple facts to our database when a conjunction is present in the sentence.

Results

To show the result obtained we will again exemplify.

Input:

"Marge is the mother of Bart , Lisa and Maggie.Homer is the father of Lisa.Patty and Selma are the sisters of Marge.Bart is the eldest son of Homer"

Output:

mother(marge, bart).
mother(marge, lisa).
mother(marge, maggie).
father(homer, lisa).
sister(patty, marge).
sister(selma, marge).
child(bart, homer).

Discussion

We first note that the output obtained correctly represents the text provided, thus the objective is attained. However, the work done presents some limits. Firstly, we are forced to provide the system with a vocabulary, the words we expect to see in the sentences. If a word is not recognized the system will ignore that sentence. We could improve this by *skipping* the word, however this could lead to parsing incorrect sentences. The vocabulary includes the name of the family members this could again be seen as a limit.

To conclude, we can say that that with a fairly simple system we managed to capture the essence of a family tree.

References

- [1] Lab on Procedural Semantics https://aicourse.r2.enst.fr/CANLP/tp_ProceduralSem.html



SD213

Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Name: Pauline ESCAVI

Italian grammar in NLP

June 29, 2023

Abstract

Abstract

This mini-study explores the challenges of incorporating Italian grammar into natural language processing (NLP) systems. The study outlines a step-by-step approach to convert intuitive grammar rules into code, focusing on determinants, verb conjugation, and noun gender and number. The results indicate successful implementation of certain aspects, but also highlight limitations and the need for further improvements in handling complex language structures.

Problem

NLP is universal and is a way for computers to understand and manipulate human languages. Therefore it is enriching to train computers with different languages, such as Italian. However, Italian is a grammatically more complex language than English, with many types of determinants and rules depending on some cases.

The problem is then how to convert all the intuitive grammar rules into code that computers will be able to understand.

Method

To solve the problem of the Italian grammar being more complex than the English one, we had to make it step by step. We first created a simple code frame similar to the ones from

SD206 labs. Then step by step we added some rules and information.

Step 1:

The first step was implementing the determinants rules. In Italian, there is different determinants corresponding to different type of nouns. The following tab explain how it works for the definite determinants:

	case A	case B	case C
masculine singular	il	l'	lo
feminine singular	la	l'	la
masculine plural	i	gli	gli
feminine plural	le	le	le

case A: the noun following the determinant starts with a consonant

case B: the noun following the determinant starts with a vowel

case C: the noun following the determinant starts with 'z' or a 's'+consonant

As we see, we have to implement predicates to deal with the different rules, to check the beginning of the noun following the determinant to have it right.

Step 2:

The second step was to differentiate verbs whether they are 3rd person singular or 3rd person plural. Once we differentiate this, we have to link the code corresponding to the already existing code about the determinant. So that if we check a simple sentence such as `il cane cammina` (the dog walks) it says it is correct.

Step 3:

The third step was to extend the step 2 by automatically check whether a verb is correctly conjugated for all persons in present. In Italian, verbs in present have specific terminations following this model:

	verbs in <i>ARE</i>	verbs in <i>ERE</i>	verbs in <i>IRE</i>
1st Person singular	-o	-o	-o
2nd Person singular	-i	-i	-i
3rd Person singular	-a	-e	-e
1st Person plural	-iamo	-iamo	-iamo
2nd Person plural	-ate	-ete	-ite
3rd Person plural	-ano	-ono	-ono

The type of verbs ARE,IRE,ERE correspond to the termination of the infinitive verb. We can then use these terminations to check the end of a word to see if it is a verb and at which person it is conjugated.

Step 4:

The last step was to automatically check for the gender and number of a noun. In Italian there is a rule to say if a noun is of a certain gender and number. It follows this model:

	termination
masculine singular	-o
feminine singular	-a
masculine plural	-i
feminine plural	-e

Therefore to check the gender and number of the word **zia** (aunt) for example we just have to look at the last letter, it is a **a** and so the word is feminine singular.

Results

The frame used from SD206 (`grammar1.pl`) allowed me to check simple sentences such as :

```
?- s([il, cane, piace, mario], []). %the dog likes Mario
true .
```

```
?- s([il, cane, piace, il], []). % the dog likes the
false.
```

Which are results that make sense.

Then I improved the basic frame. The first step was quite easy to implement and gave satisfying results. You can run the file `grammar2.pl` and check all the different combinations. For example we have:

```
?- s([il, cane, pensa], []).
true .
```

```
?- s([lo, cane, pensa], []).
false.
```

```
?- s([i, cane, pensa], []).
false.
```

```
?- s([la, sorella, pensa], []).
true .
```

```
?- s([la, zia, pensa], []).
true .
```

All these tests are successful and the code is only using predicates that check the beginning of the following noun.

The step 2 (check `grammar3.pl`) was also a success because it was really simple to add attributes like `singular` or `plural` to the verbs rules so we could check for coherence. However, the approach from this step is time consuming because we have to manually enter the verbs. Here are some examples of the tests I made:

```
?- s([il,cane,pensano], []). %the dog think
false.
```

```
?- s([il,cane,pensa], []). % the dog thinks
true .
```

The results of step 3 are much more mixed. I coded this step on a separate file (`check_verb.pl`). The goal by separating the grammar file from this file was to concentrate more on the predicates and content of `check_verb`. We can then put a word that we want to know if it is a

verb or not, and as an output it says `true` if it is verb and it gives the person and number it was conjugated to. For example we can try:

```
?- v(Person, Number, parliamo,_,_). % means: we talk
Person = first,
Number = plural .
```

```
?- v(Person, Number, parliar,_,_). %means nothing
false.
```

The code separately works however I was not able to integrate properly the code in the already existing code. I tried to integrate it in the file `grammar4.pl` but tests were not successful because of many errors.

Finally the last step has almost the same conclusion as for the last step. I coded on a separate file predicates to check the end of nouns so we could deduce what gender and number they have. The code (`gender_check.pl`) itself worked and here are examples of tests:

```
?- n([zia],Number, Gender).% zia = aunt
Number = singular,
Gender = feminine .
```

```
?- n([zie],Number, Gender). % zie = aunts
Number = plural,
Gender = feminine .
```

However, once again I could not integrate the code in the previous grammar file. I tried to integrate it in (`grammar5.pl`) but tests were not concluding.

Discussion

The predicates checking for verb conjugation or for noun gender and number were successful when put separately which was a good news. However, these predicates have many limitations. The first one is that for the verb checking, predicates work by checking the termination of the word given. And for example if we take a noun that end with an `a` like `zia` (aunt) the predicate will say that it is a verb conjugated at the third person singular, which is obviously false.

The second problem is that the two checking predicates only work by checking the end of a word and does not have any knowledge of what differentiate a verb from a noun and this question is much more complex than what we could think. Therefore it is easy to test the code with tricky sentences and words and see the code get wrong.

To conclude there are many way to improve the code: adding lexicon, adding grammar rules (different tenses, preposition + determinant fusion, object complement rules, etc...)

Bibliography

SD206 labs: <https://aicourse.r2.enst.fr/LKR>

SD213 labs: <https://aicourse.r2.enst.fr/CANLP>

Verb Conjugation example:

<https://www.rangakrish.com/index.php/2019/09/29/generating-verb-conjugations/>



SD213

Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Name: Roel George, Thang Bui Doan

Vietnamese Parser

July 1, 2023

Abstract

Cognitive Natural Language Processing plays a significant role in extracting meaning and structure from human language. One of the fundamental tasks in NLP is syntactic parsing, which involves analyzing the grammatical structure of sentences. While several parsers exist for widely spoken languages such as English and French, there is a need to develop parsers specifically tailored to handle the unique linguistic characteristics of less-resourced languages like Vietnamese.

Problem

So far there have been many new developments in the field of NLP and have seen various ways to parse texts in various languages. But most of the parsers are in more commonly used languages like English and French. Thus, development and therefore its extension to various other fields are more prominent and well-known. We observed this difference in the advancement of languages like English and not as commonly used languages like Vietnamese and decided to come up with a parser for Vietnamese.

Vietnamese grammar follows the “SVO” (Subject-Verb-Object) structure, just like English and French grammar. But it also has a few different points compared to those mentioned languages. Vietnamese does not have tenses and determinants. There is no difference in verb conjugation neither between genders nor between singular and plural nouns. But there are restrictions and preferences on how words are used, so to parse Vietnamese sentences completely, restrictive rules

may be applied through argumentation. In this project, we try to simplify the problem by using common sentence structures, which work for all cases of verbs and nouns.

Another problem in Vietnamese is that each word can be formed from 2 elements. For example, The Vietnamese word for "sometimes" is “thỉnh thoảng”. If we use a regular parser, the word will be split into 2 elements: “thỉnh” and “thoảng”, which does not mean anything, the standard parser would fail to recognize the structure of the whole sentence. Moreover, one element can mean something but adding another element next to it would result in a complete change in the meaning of the sentence. For eg: "dễ" is Vietnamese for "easy" but "dễ thương" means "cute". This can create an ambiguity while parsing because during word segmentation, the word "dễ thương" will be split to "dễ" and "thương" which makes capturing the meaning of the word much harder.

Method

Since the Prolog program cannot operate well with Vietnamese accents, we decided to implement our solution in Python instead. And to solve the 2 elements problems we mentioned above, instead of only finding word structure in one element, every 2 consecutive elements were also tested.

We use the Bottom-up Chart Parsing approach in our solution: Bottom-up chart parsing is a parsing technique used in natural language processing (NLP) to analyze the structure of a sentence based on a given grammar. Different from the normal parser, the chart parser memorizes partial results obtained during parsing to avoid making the exact computations endlessly.

The code runs by understanding grammar and rules by reading the PSG3.txt file and then using this knowledge along with the above-mentioned Bottom-up parsing approach to parse the given input Vietnamese text.

Here is a small example of what constitutes the grammar for the implementation:

Grammar	Lexicon
⋮	⋮
NP -> N	N -> me
NP -> N Adj	N -> chi
NP -> N PP	N -> tôi
NP -> N Adj PP	Adj -> xanh
NP -> N NP	Adv -> thỉnh thoảng
VP -> V NP	V -> yêu
VP -> V	V -> nhìn
VP -> V PP	P -> trên
PP -> P NP	P -> cùng
⋮	⋮

Results

We success in parsing the sentence with the “1-element only” word.

We take the sentence “chi tôi ăn com” as an example

Adding edge: (0, 1, 1, 'N', ('chi'), (,))
RI Adding edge: (0, 1, 1, 'NP', ('N'), (0,))
RI Adding edge: (0, 1, 1, 'NP', ('N', 'Adj'), (0,))
RI Adding edge: (0, 1, 1, 'NP', ('N', 'PP'), (0,))
RI Adding edge: (0, 1, 1, 'NP', ('N', 'Adj', 'PP'), (0,))
RI Adding edge: (0, 1, 1, 'NP', ('N', 'NP'), (0,))
RI Adding edge: (0, 1, 1, 'S', ('NP', 'VP'), (1,))
Adding edge: (1, 2, 1, 'N', ('tôi'), (,))
RI Adding edge: (1, 2, 1, 'NP', ('N'), (7,))
RI Adding edge: (1, 2, 1, 'NP', ('N', 'Adj'), (7,))
RI Adding edge: (1, 2, 1, 'NP', ('N', 'PP'), (7,))
RI Adding edge: (1, 2, 1, 'NP', ('N', 'Adj', 'PP'), (7,))
RI Adding edge: (1, 2, 1, 'NP', ('N', 'NP'), (7,))
FR Adding edge: (0, 2, 2, 'NP', ('N', 'NP'), (0, 8))
RI Adding edge: (1, 2, 1, 'S', ('NP', 'VP'), (8,))
RI Adding edge: (0, 2, 1, 'S', ('NP', 'VP'), (13,))
Adding edge: (2, 3, 1, 'V', ('ăn'), (,))
RI Adding edge: (2, 3, 1, 'VP', ('V', 'NP'), (16,))
RI Adding edge: (2, 3, 1, 'VP', ('V'), (16,))
RI Adding edge: (2, 3, 1, 'VP', ('V', 'PP'), (16,))
FR Adding edge: (1, 3, 2, 'S', ('NP', 'VP'), (8, 18))
FR Adding edge: (0, 3, 2, 'S', ('NP', 'VP'), (13, 18))
Adding edge: (3, 4, 1, 'N', ('com'), (,))
RI Adding edge: (3, 4, 1, 'NP', ('N'), (22,))
FR Adding edge: (2, 4, 2, 'VP', ('V', 'NP'), (16, 23))
FR Adding edge: (1, 4, 2, 'S', ('NP', 'VP'), (8, 24))
FR Adding edge: (0, 4, 2, 'S', ('NP', 'VP'), (13, 24))

Right after the first word is recognized, the potential structures are immediately added as inactive edges, thus recognizing the whole sentence structure. The final parsing result:

[S [NP [N tôi]] [VP [V ăn] [NP [N com]]]]

Discussion



Our initial objective was to successfully parse sentences with a “2-element” word, such as the word “thành thạo”. However, we did not accomplish and were only able to parse words with 1 element. Our future plan is to continue this project and realize this goal, also expand the limited vocabulary we currently used.

Moreover, Vietnamese has restriction and preference in using vocabulary in some specific grammar structure. Our group can expand this project to work with those restrictions. Once the final version of the parser is completed, It can also be extended to complete other Statistical NLP tasks like Machine Translation, and Sentiment Analysis as well.

Bibliography

- [Link to GitHub repository- Click here](#)

- Cognitive approach to NLP- J-L. Dessalles
- Syntactic Parsing- J-L. Dessalles
- Charty in Python by Damir Cavar
- Bottom up Parsing

 	<p style="text-align: right;">June 2023</p> <p style="text-align: center;">SD-213</p> <p style="text-align: center;">Cognitive Approach to Natural Language Processing</p> <p style="text-align: center;">Micro-study</p> <p style="text-align: right;">teaching.dessalles.fr/CANLP</p>
---	--

Name: Olivier JASSON

Recognizing punctuation

Abstract

This project aims at augmenting the level of recognition of a sentence to include punctuation, and the different syntaxes it refers to.

Problem

In this project, I want to make the recognition of true sentences by the computer better. In order to do so, I'll add the recognition of punctuation, both commas and points, and the different syntaxes that use this symbols. At the end, it would be nice to be able to recognize several structures, such as a question, a coordination, a precision made between commas, a list, and an introduction to a sentence (just like for this one).

Method

The first thing that defines a sentence is a number of words which ends with a point. The first step is then to add the recognition of the different points.

```
pt --> [.]
pt --> [!].
pi --> [?].
```

```
s --> as.      % Assertive sentence
s --> q.       % Question

q --> qs, pi.  % Question sentence, questin mark
qs --> ia, iv. % Interrogative adverb, inversion verb subject
qs --> ax, vp. % direct question with auxiliary : Do you like it ?
qs --> ax, np. % direct question : Are you tired ?
qs --> as, ei. % assertive sentence, ending iterrogation : This is it, isn't it ?
iv --> ax, np. % auxiliary, noun phrase : What is this ?
iv --> ax, vp. % auxiliary, verb phrase : What do you want ?
ei --> cm, er. % comma, aux+pronoun
er --> ax, pr. % auxiliary, pronoun
```

The question mark is of course the one that brings most trouble, as we need to identify all the kinds of questions. A flaw of this method can already begin to be seen : I add a lot of different names and notations, and the glossary will also need a few more different categories.

After this, we deal with the comma. The symbol itself is used in prolog to separate arguments in a list, and that's a problem for me. That's why I decided to replace it by a semicolon in the code, and I assumed that there is a way of translating. The semicolon itself is not a huge loss, as it works as a special kind of comma. On the same track, this parser is made to recognise words or symbols that are separated by spaces. In the test, we need the comma to be separated from the text. I obtained the following rules :

```

as --> sp, pt.      % sentence, point
sp --> np, vp.      % classique sentence : I eat chocolate
sp --> sp, cs.      % sentence, coordinated sentence
cs --> co, sp.      % coordination, coordinated sentence
sp --> in, sp.      % introduction, sentence
in --> pr, npc.    % preposition, noun phrase + comma
npc --> np, cm.

co --> cc.          % coordination adverb
co --> cm, cc.      % comma, coordination adverb
co --> cm.          % comma

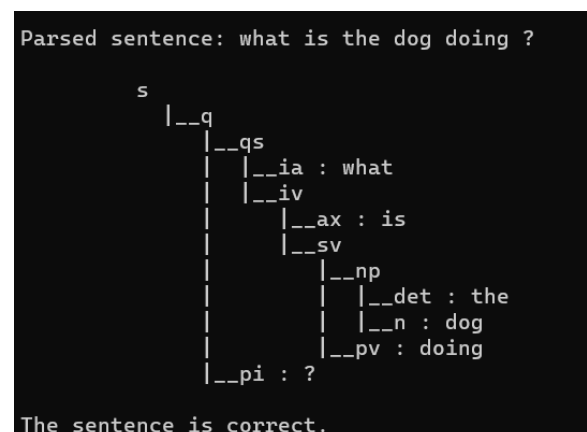
np --> np, cn.      % noun phrase, coordinated noun phrase : the cat and the mouse
cn --> co, np.      % coordination, second noun phrase

np --> det, n.      % Simple noun phrase
np --> np, pp.      % Noun phrase, prepositional phrase
np --> det, an.     % det, adjectives + noun
an --> ads, anp.    % adjectives, rest of the np
anp --> n.
anp --> n, pp.
ads --> ad.         % adjective
ads --> ad, co.     % list of adjectives
np --> np, i.       % noun phrase, incised precision
i --> il, cm.       % incise, comma
il --> cm, np.     % comma, incise
np --> [kirk].

```

I dealt with lists, coordination, precisions and introductions.

Results

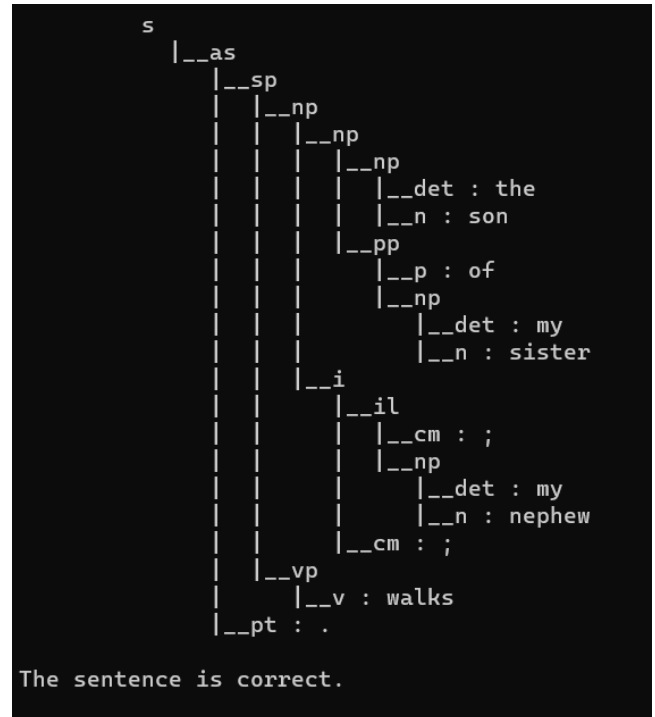
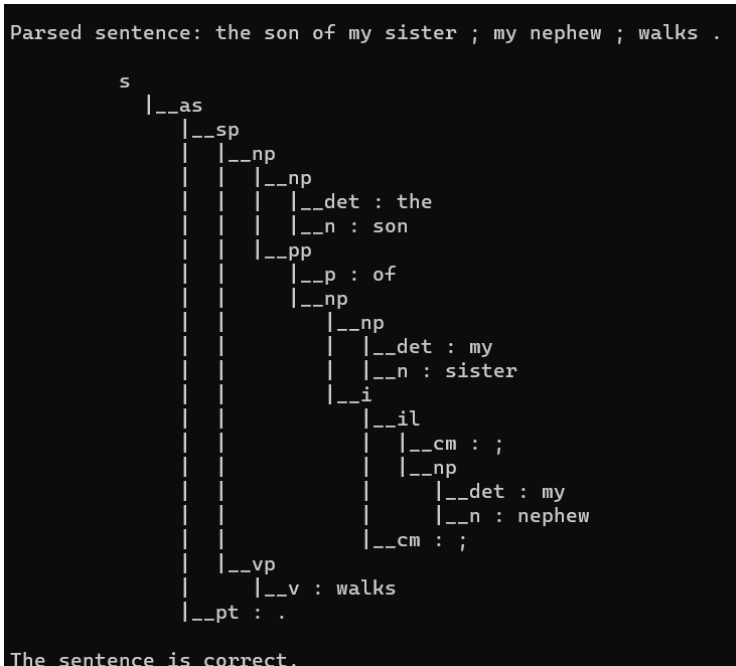


I can now recognise most of the syntaxes linked with punctuation.

Discussion

The program works pretty well, and I can recognize what I wanted. Yet, I had to implement many new labels, and the further we go, the more complicated and specific the program will get. At the end, it is as good as having a rule for each word.

Also, there are still inconsistencies in the recognition, as shown in this example.



The original sentence is : “The son of my sister, my nephew, walks.”. The parser can see two ways of recognising this sentence, and both are grammatically strictly correct. This parser still needs more advanced Procedural Semantics

Bibliography

This work is based on the workshop designed by Jean-Louis Dessalles. His site is accessible here : <https://aicourse.r2.enst.fr/CANLP>.



Cognitive Approaches to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Students: Antoine Andurao and Lisa Lloret

Procedural semantics in Game of thrones

June 29, 2023



Abstract

Abstract This project focuses on creating a mini-knowledge base for Game of Thrones, exploring the intricate relationships between its characters in a fantasy setting. By analyzing this complexity, we seek to gain deeper insights into the dynamics of the Game of Thrones universe.

1 Introduction

This is a project within the context of the Cognitive Approach to Natural Language Processing course (SD213) taught by Professor Jean-Louis Dessales. Our aim is to expand on the lab work done on Procedural semantics by creating a mini-knowledge base focused on a specific subject. We have chosen the domain of Game of Thrones, an American fantasy drama television series produced by HBO. The show is based on the book series called A Song of Ice and Fire (cf. [1]), written by George R. R. Martin, with the first book titled A Game of Thrones. Set in a fantasy world filled with creatures and characters, each episode presents a complex web of relationships that we seek to understand. Your objective in this game is to comprehend the intricate connections that exist between the characters.

2 Objective

Through this project, our aim is to gain a deeper understanding of the intricate relationships that exist among the characters in the series Game of Thrones. Indeed, with such a complex narrative, it becomes rather challenging to discern the affiliations between individuals, ascertain their parentage, and identify who stands as ally or enemy to whom.

3 Project's steps

Augmenting the World Constraints with new predicates

The primary step involved in this endeavor was to establish new predicates in order to enhance the understanding of the Game of Thrones world, specifically the relationships between the characters. To accomplish this, we introduced additional characters and implemented more precise predicates, such as gender, relationships, and genealogical affiliations, for a comprehensive depiction.

Updating the grammar

Subsequently, we proceeded to enrich the existing grammar. To achieve this, we commenced by incorporating new nouns, adjectives representing families, and verbs into the framework.

Testing and going further

Once we've worked out the details of how this first version works, we can continue to expand the knowledge base and grammar.

4 Method

Data base

For the database of this project, we choose the specific domain of the Game of thrones. One of the primary factors influencing this decision is the abundance of information available on the internet regarding this television series. Additionally, the data pertaining to Game of Thrones is highly modular, allowing us to incrementally enhance the complexity of the database as the project progresses. As part of this endeavor, we have chosen to incorporate details such as the

gender of each character and their relationships (parent, ally, enemy, etc.) (cf. [2]). To accomplish this, we discovered a suitable graphic online from which we extracted the required information.

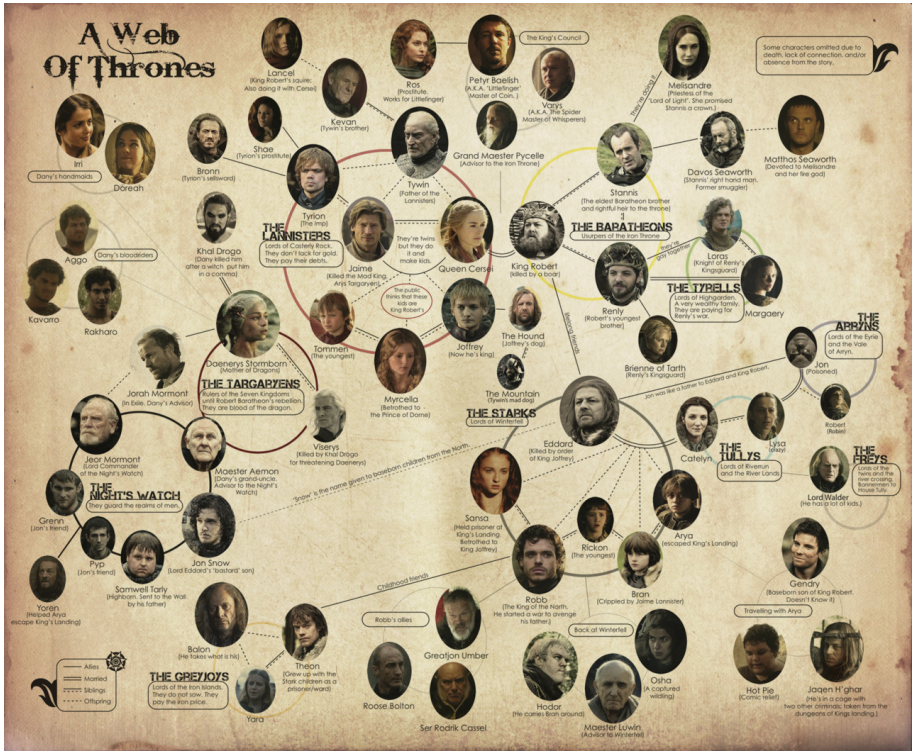


Figure 1: Tree of relationship between characters

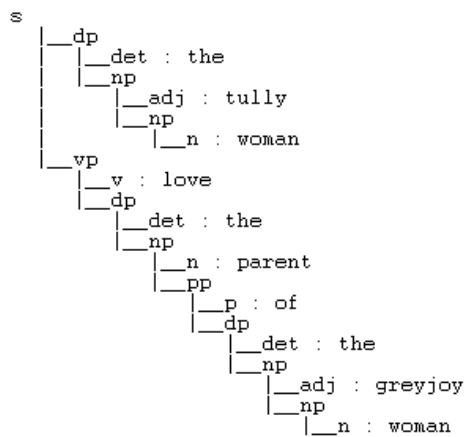
We have created another file, named ' ' in order to organize our grammar world.

5 Results

We managed to have a rich enough grammar to describe the character by their gender and relationships to other characters (families, love affairs), and to parse those descriptions to actual characters.

As shown below, the sentence "the tully woman loves the parent of the greyjoy woman" could be parsed to love(catelyn, balon). Yet, the predicates in the world constraints prevent this syntactilly correct sentence to be considered true.

```
Syntactically correct
[gloss:love,num:sing,pers:3,subj:dp([gloss:woman,num:sing]),cpl:[dp([gloss:parent,num:sing])]]
```

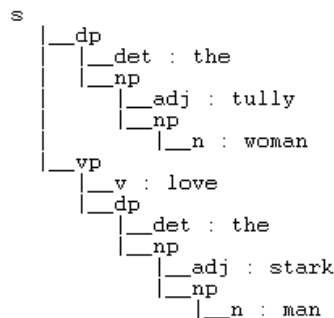


```
===> No, Catelyn and Balon don't love each other!
```

```
...> love(Catelyn,Balon)
--> love(Catelyn,Balon)
this sentence makes sense
type [:] to see alternatives... ■
```

Figure 2: First results

```
Syntactically correct
[gloss:love,num:sing,pers:3,subj:dp([gloss:woman,num:sing]),cpl:[dp([gloss:man,num:sing])]]
```



```
Yes they do !
```

```
...> love(Catelyn,Eddard)
--> love(Catelyn,Eddard)
this sentence makes sense
type [:] to see alternatives...
```

Figure 3: Other results

In the third figure, for the 'love' predicate, you can see the '!' symbol to stop backtracking. This allows the parser to interpret only the adequate love relationship. If the love relationship actually exists, then only `love(A,B) :- "Yes they do"` will be executed.

6 Discussion

Comparison with expectations

Despite having still lots to do, we've been able to set up a knowledge base representing part of characters and the relationships in Game of Thrones, and adapt the grammar from the lab to

interpret this knowledge.

Main drawback

Throughout this project, we encountered several challenges. The most significant obstacle that we were unable to overcome is the lack of recognition for proper nouns (PN) within prepositional phrases (PP). Although it seemed to us that the DCG parser was already designed for this, we were unable to get sentences such as "Robert's wife" to interpret as wife(Cersei, Robert), whereas "the king's wife" did.

Perspectives

As a further improvement, we'd like to dive deeper in the DCG parser and grammar, in order to make it possible to parse such sentences. We are also thinking of augmenting again our knowledge base with more characters, more precise and intricate relationship such as betrayals, and maybe a notion of time (e.g. "Cersei loved Jaime") because the relationship evolve other time. Hence, the domain of natural language processing can be expanded to encompass additional factors, enabling the generation and comprehension of more intricate dialogues. An intriguing approach would involve the utilization of the Conflict-Abduction-Negation (CAN) concept to explore the Game of Thrones scenario and facilitate an understanding of the relationships among the numerous characters.

7 Conclusion

All in all, we derived immense pleasure from undertaking this project, which revolved around a highly engaging theme. Moreover, through this practical endeavor, we acquired the ability to comprehend and modify the grammar to generate more complex sentences. Consequently, we deepened our understanding of procedural semantics, albeit not accomplishing all of our initially set objectives. Nevertheless, we take pride in our final submission.

8 Acknowledgments

We would like to express our heartfelt gratitude to our supervising professor, Mr. Jean-Louis Dessalles, for his valuable time and unwavering dedication. Throughout this period, he imparted to us his profound passion for natural language processing and cognitive sciences.

References

- [1] "Game of Thrones". In: *Wikipedia* (). DOI: https://en.wikipedia.org/wiki/Game_of_Thrones.
- [2] "Tree Relationship". In: *Blogspot* (). DOI: https://2.bp.blogspot.com/-pASmQ80S1so/Ub4zwV50fTI/AAAAAAAAODA/EVDsPTv_rsI/s1600/Game-Of-Thrones-Family-Tree.jpg.



6/29/2023

SD-213

Cognitive Approach to Natural Language Processing

Micro-study

teaching.dessalles.fr/CANLP

Name: Albert Noubissi, Léon Sillano

Building grammar for football player's position

Abstract

This project aims to develop a grammar for footballer's player. The goal is to ask the programme with English language in order to get information on the position of footballer according to a certain data, and to change this data with natural language.

Problem

We have data on the position of football players of the team PSG, and we want to get this information by asking sentences to the program. For instance, we want to know which player is to the left of Mbappe, or which player is ahead of Verratti, another player of the PSG team. Then we want to change this data, we want to move players on the field relatively to other players.

Method

For this project, we used prolog and the code of Procedural Semantics of the teacher Jean-Louis Dessalles and we extended the grammar by modifying `sem_grammar.pl` and we added a new file `sem_Football.pl` in order to add the semantic world of Football.

In `sem_football.pl` we first added the data of PSG players. The data is represented like this:

`player(mbappe, forward, (2,1))` which means the player Mbappe, who is a Forward, is placed in the position (2,1) on the grid. The explanation of the grid is given in figure 1.

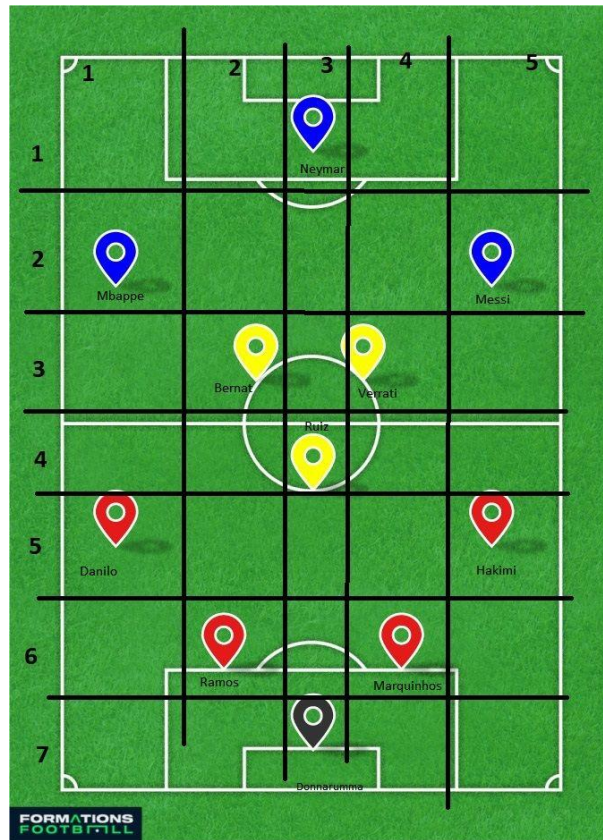


figure 1: Football's Grid

Moreover, we added new words in the grammar, which are the different proper nouns of the players and as a matter of fact we have this examples:

```
pn([gloss:neymar, num:sing], player(neymar,_,_), pn(neymar)) --> ['Neymar'].
pn([gloss:mbappe, num:sing], player(mbappe,_,_), pn(mbappe)) --> ['Mbappe'].
```

We also added the nouns of all the positions which are : forward, midfielder, defender, goalkeeper.

```
n([gloss:N, num:sing], P,n(N)) --> [N], {member(N, [forward,
midfielder, defender, goalkeeper]), P =.. [N, _, _]}.
```

We divided this project into two steps.

- The first one aims to get the relative position of players. For instance, we would ask “to the left of Mbappe”, and the programme would answer all the players who have the same role on the fields as Mbappe and who are to the left of Mbappe on the grid.

To do that, we first added new words which are `left`, `right`, `above` and `behind`.

Then we created predicates `left`, `right`, `above` and `behind`. In this case, for instance, for `left`, player A is to the left of player B if A has the position (x1, y1) and B has the position (x2,y2) such that $y1 < y2$ and the two players have the same position. For instance, we can compare Messi and Mbappe because they are two forwards, but we

cannot compare Verratti and Mbappe because Verratti is a midfielder and Mbappe is a forward.

- The second step is to add a new verb, which is `move`. We can use this verb to express the claim of changing the data. For instance, we want to implement : Mbappe moves to the right of Messi” and the program will change the data. To avoid confusion, in that case, “to the right of Messi” means in that case the position to the right and just near to that player. If Messi has position (2,5), to the right of Messi will be (2,6). Thus, we added new predicates for right, left, above and behind in order to handle this need. One `right` predicate will be for getting all the players to the right of the current player, and another predicate `right` will be used for getting the position next to the right of the current player.

Then, we added the predicate `move` which uses `retract` and `assert` in order to change the data. If the information given in that predicate is correct, then the program will retract the predicate `player(_,_,Anc. Loc)` containing the ancient location and add the predicate `player(_,_,New loc)` containing the new location of the player.

For instance, “Mbappe moves to the right of Neymar” will retract the data:

```
player(mbappe, forward, (2, 1))
```

and add the predicate:

```
player(mbappe, forward, (1, 4))
```

Move is based on the semantic linking proposed by the program made by Jean Louis Dessalles. `move(Pos, Player0)` will be linked with `right(Pos, Player1)` and `player(Player0)`.

Results

As mentioned above we will present the results of each part of our work. First of all as far as the relative positions of the players are concerned we got pretty good results. the program was able to understand and query phrases like ‘to the right of *Player*’. In this case the player can be either a forward , midfielder, defender or a goalkeeper.

Figure2: we ask a relative position “to the right of Mbappe”

```
?- consult(sem_main0).
true.

?- go(pp).

Phrase > to the right of Mbappe
linking right(_8614,_8616) with player(mbappe,_8688,_8690)
executing player(mbappe,_8688,_8690)
...> player(mbappe,forward,(2,1))

Syntactically correct
[gloss:to]

pp
|__p : to
|__dp
|__det : the
|__np
|__n : right
|__pp
|__p : of
|__dp
|__pn : mbappe

executing right(_8614,mbappe)
...> right(player(neyanmar,forward,(1,3)),mbappe)
right(player(neyanmar,forward,(1,3)),mbappe)
```

Figure2 bis: the answer of the request. We get the players Neymar and Messi.

```
executing right(_8614,mbappe)
...> right(player(neymar,forward,(1,3)),mbappe)
right(player(neymar,forward,(1,3)),mbappe)
this sentence makes sense
...> right(player(messi,forward,(2,5)),mbappe)
right(player(messi,forward,(2,5)),mbappe)
this sentence makes sense
```

In figure 2 we can observe that by querying ‘to the right of Mbappe’ we got Neymar and Messi with their respective positions who are to the right and play the same role as Mbappe (Forward).

Following the same logic we can also try to know the relative positions of players ahead of other players. The next figure shows how we retrieved the positions of players ahead of Ruiz (midfielder) and we got Verratti and Bernat who are midfielders too standing ahead of Ruiz.

```
Phrase > to the ahead of Ruiz
linking ahead(_25818,_25820) with player(ruiz,_25892,_25894)
executing player(ruiz,_25892,_25894)
...> player(ruiz,midfielder,(4,3))

executing ahead(_25818,ruiz)
...> ahead(player(bernat,midfielder,(3,2)),ruiz)
ahead(player(bernat,midfielder,(3,2)),ruiz)
this sentence makes sense
...> ahead(player(verratti,midfielder,(3,4)),ruiz)
ahead(player(verratti,midfielder,(3,4)),ruiz)
this sentence makes sense
```

figure 3: find players ahead of Ruiz.

As you may have noticed we can keep querying phrases like ‘to the left of Player’ or ‘behind of Player’, The program will understand and retrieve the corresponding relative positions.

The second part of our project was to be able to change the position of a player on the field relative to other players already present. The query phrase is actually ‘Player1 move to the right of Player2’. with Player1= Mbappe and Player2=Neymar we got ‘Mbappe move to the right of Neymar’. The desired result is the one in figure 5.

```

Sentence > Mbappe move to the right of Neymar
linking move(_38194,_38196) with right(_38304,_38306)
executing right(_38196,_38306)

Phrase > to the left of Neymar
linking left(_25854,_25856) with player(neymar,_25928,_25930)
executing player(neymar,_25928,_25930)
...> player(neymar,forward,(1,3))
executing left(_25854,(1,3))
linking left(_26274,_26276) with player(neymar,_26348,_26350)
executing player(neymar,_26348,_26350)
...> player(neymar,forward,(1,3))
executing player(neymar,_26276,_26350)
...> player(neymar,forward,(1,3))
executing player(neymar,_26348,_26276)
...> player(neymar,forward,(1,3))
false.

```

figure 4: moving Mbappe to the right of Neymar.

In figure 4 we queried ‘Mbappe move to the right of Neymar’ and we asked the player which players stand to the right of Neymar: we got Messi and Mbappe which was the desired result. Right after we asked the program ‘to the left of Neymar’ and the result was ‘false’ which is correct too since there are no players standing to the left of Neymar who are forwarders.

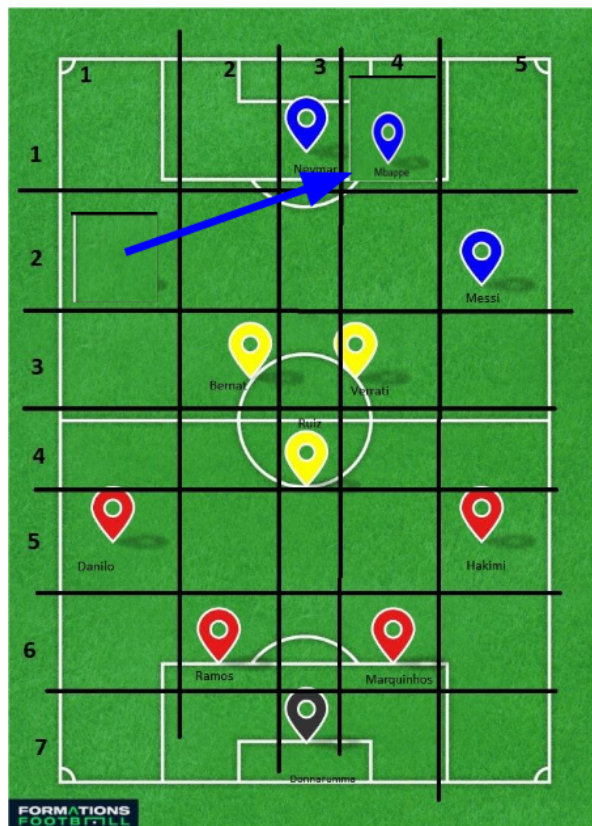


figure 5: Desired result moving Mbappe to the right of Neymar

Discussion

In this project, we wanted to extend the notion of position to footballers players, by redefining the notion of relative right or relative left, which now includes the notion of position. The second part of the project about the function move was quite tough, because it demands a deep comprehension of the prolog code. For students like us for this level on logic programming language, the second step was difficult, and needed a lot of time.

We can improve our model by adding new conditions for moving a player, or even changing a player with its substitute. There are also different ways to change the data, without using retract and assert. The use of these functions was quite difficult. We think that we can do the task more simply.

Bibliography

- Procedural semantic practical work by Jean-Louis Dessalles:
<https://aicourse.r2.enst.fr/CANLP>
- PSG footballer's player in the position 4-3-3 drawn by the website Formations football: <https://www.formationenfootball.fr/>