Advanced Programming and Algorithms
CKY Algorithm (Cocke–Kasami–Younger)

June 15, 2025

Contents

1 Submitted Files

- `cky.py` – Implementation of the CKY algorithm
- `test.py` – Test cases for different grammars and words
- `gramatiques.py` – Definition of grammars and words
- `extensio_1.py` – CFG to CNF transformation
- `extensio2.py` – Probabilistic CKY implementation
- `informe.pdf` – Practice report, this document

2 CKY Algorithm

Before starting with the programming of our code, we need to keep in mind how the CKY algorithm operates.

Initially, in class we studied a version that used a square matrix of dimensions $n(n+1)/2$, where n is the length of the input string, and where the indices were traversed as **diagonal** and **column**, in a staggered structure.

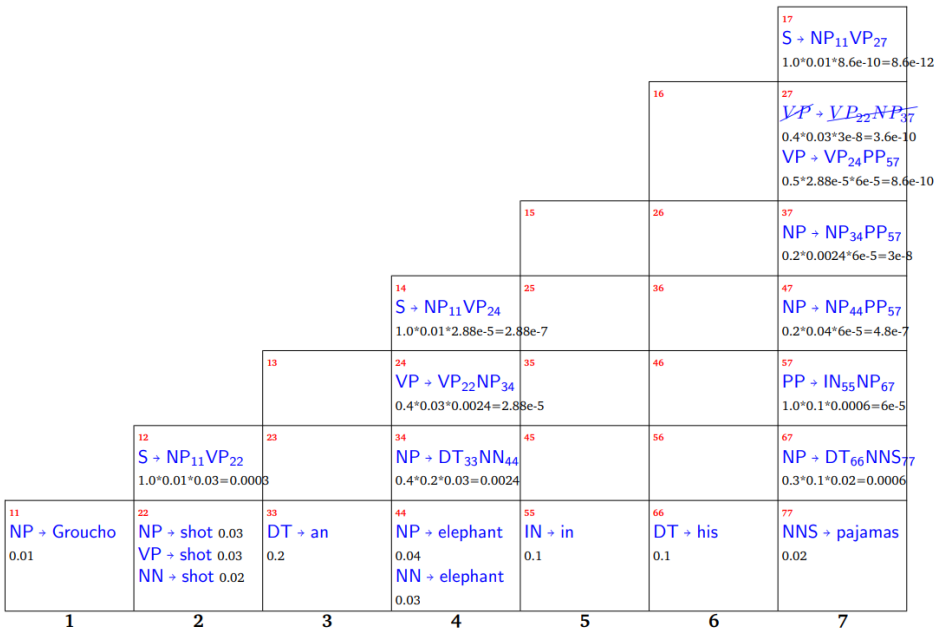

Figure 1: Triangular matrix of the CKY algorithm seen in class (PLH)

To simplify the implementation, initially we opted to use a square matrix accessing through **diagonal** and **column** indices, following step by step the algorithm seen in class. However, we observed that this approach was inefficient in terms of memory, since a large number of unnecessary cells that were never used were stored (almost half).

For this reason, we finally decided to implement an optimized version that only uses the **upper half of the matrix**, that is, an **upper triangular** structure. This optimization has allowed us to significantly reduce memory usage, while maintaining a clear and efficient structure for programming the algorithm. However, this forces us to change the interpretation of the indices and, consequently, slightly modify the representation of the algorithm.

Thus, in the final implementation we start by filling the first row with terminal productions. During this process we identify which non-terminals can directly generate each word through rules of the type $A \rightarrow a$. The second phase consists of building the table bottom-up, processing substrings of increasing length in the process. For each substring, the algorithm tries all possible divisions into two parts and consults the already calculated cells. When both parts contain non-terminals, the algorithm checks if there exists any rule $A \rightarrow BC$ that combines them. We can imagine the indices as [length-1][start], where length is the length of the substring and start is the start index of the substring in the sentence. This phase continues until the entire table is traversed.

We can imagine the table with indices as follows:

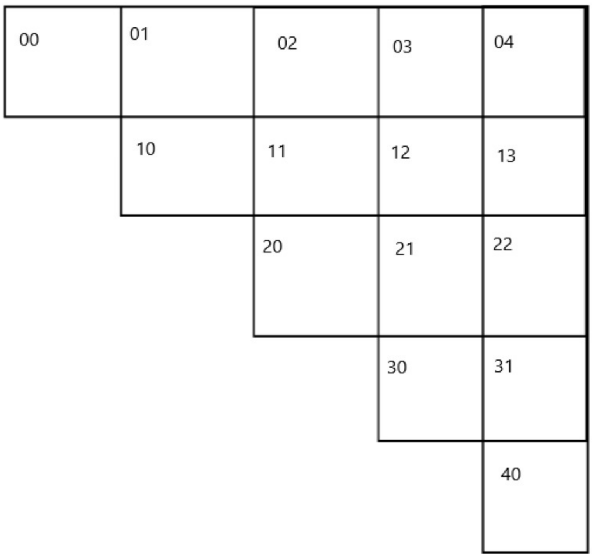

Figure 2: Upper triangular matrix of our CKY algorithm

Where for each cell we will check in order the cells of the same diagonal (from the farthest cell to the nearest) and of the same column (from the nearest cell to the farthest). Note that the cells are shifted to the right for better understanding of the algorithm.

Results Validation

```
S -> A B, C D, C B, S S
A -> B C, a
B -> S C, b
C -> D D, b
D -> B A

Sentence: 'bab' -> False
Sentence: 'abab' -> True

S -> A B,
A -> a,
B -> b

Sentence: '' -> True
```

To show the functioning of the algorithm, we present two small examples of interesting outputs, although the file structure will be shown in the next section and the rest of the results are found in annexes.

In the first example, the algorithm correctly recognizes the sentence `abab` because it is possible to derive it from the grammar: $S \Rightarrow A\,B \Rightarrow a\,B \Rightarrow a\,b$. And since the production $S \Rightarrow S\,S$ also exists, we can obtain `abab`. However, we see that other strings like `bab` are not derivable with any sequence of productions, and therefore the result is `False`. In the second example, the base case of the empty string is highlighted, and since S can generate it, the result is `True`.

3 Explanation of the Developed Code

3.1 cky.py

This file contains the `Gramatica` class, which allows representing a grammar in Chomsky Normal Form (CNF) and applying the CKY algorithm to determine if a word belongs to the language generated by the grammar.

The main methods are:

- `__init__`: initializes the grammar and preprocesses binary rules.
- `algorisme_cky(frase)`: implements the classic dynamic CKY algorithm described earlier.
- `_preprocessar_regles_binaries`: improves efficiency by searching and storing all binary productions (non-terminals).
- `_comprovar_derivacio_buida`: checks if the empty string is derivable ($S \to \epsilon$).

The program builds a triangular table where each cell contains the non-terminals that can generate the corresponding substring. At the end, it checks if the initial symbol is present in cell `[0][n-1]`.

3.2 test.py

The `test.py` file has as its main objective to facilitate the verification of the functioning of the CKY algorithm and its extensions through a set of interactive tests. This script imports different types of grammars (with and without Chomsky Normal Form and probabilistic) and passes them through different versions of the CKY algorithm to verify if they can derive certain sentences. It also includes a series of automated tests for grammars given in the statement (G1 and G2) and others defined by us. It allows validating the correct functioning of the CKY algorithm in multiple cases (accepted and non-accepted words).

The file includes various test functions. The `test_cky` function applies the CKY algorithm on simple grammars and shows if the provided sentences can be derived. The `test_fnc` function transforms these grammars to Chomsky Normal Form (CNF) and, if we indicate it, also applies the CKY algorithm to the already transformed grammar. This allows checking if the CNF conversion process has been correct and compatible with CKY. On the other hand, the `test_pcky` function works with probabilistic grammars and applies the probabilistic CKY algorithm (PCKY), also showing the most probable production tree in case the sentence is accepted.

The program incorporates an interactive menu that is shown when the file is executed. This menu allows choosing between different types of tests (basic CKY, CNF conversion, CKY with CNF, PCKY or all tests together) and visualizing the results in the console. In this way, a practical tool is provided for testing sentence recognition with different configurations of grammars and analyzers.

In the **annex** you can find some examples of the outputs that the program has returned for some of the predefined examples.

3.3 gramatiques.py

The `gramatiques.py` file provides the set of grammars and sentence examples (both CKY and PCKY) to be able to do the tests with `test.py`

Examples extracted from PLH

In addition to the initially proposed examples, we have selected a series of cases extracted from the Human Language Preprocessing course, where we have also worked with this algorithm. Specifically, these are sentences from exams from 2023 and 2024, with their corresponding morphological rules. These examples are more realistic and interpretative, which makes them especially suitable for testing the true potential of the algorithm applied to practical situations, since we can perform complex tests and easily check its behavior.

CKY Grammar Representation Format

The grammar is represented as a Python dictionary:

```
{
    "S": [["A", "B"], ["a"]],
    "A": [["B", "C"], ["a"]],
    "B": [["b"]]
}
```

Each key is a non-terminal, and each value is a list of productions. Each production is a list of terminal or non-terminal symbols. It is assumed that the grammars are already in CNF, but if they are not, we can use the `GramaticaFNC()` class.

We have chosen dictionaries as the data structure to represent grammars, because for each non-terminal it allows us to easily access each of its productions. In the case of inner lists, initially we simply had text strings ["AB", "a"], although in order to use sentences in natural language it was considered to use lists as the standard format.

Regarding lists, we have selected them over tuples since we need a mutable structure to be able to convert grammars to Chomsky Normal Form (CNF).

The grammar for natural language sentences is exactly the same, but it must be taken into account that when using the methods of the created classes, lists of strings will have to be used instead of strings directly (["good","day"]), although due to string indexing we can save this step for simplicity when dealing with simple characters ("ab").

PCKY Grammar Representation Format

In the case of probabilistic grammars, the grammar is also represented as a Python dictionary, but now the values are tuples:

```
{
    'S': [(['A', 'B'], 0.6), (['B', 'A'], 0.4)],
    'A': [(['a'], 0.7), (['A', 'A'], 0.3)],
    'B': [(['b'], 1.0)]
}
```

To represent the production-probability pairs we have chosen tuples, which allow us to maintain well-structured productions thanks to their immutability. Each production is associated with a probability, therefore, if we want to transform the grammar to CNF, we will have to eliminate the old productions and substitute them with the new ones with their corresponding probabilities.

4 Extensions Implemented

4.1 extensio_1.py: CFG to CNF Transformation

This module converts an arbitrary grammar in general form to Chomsky Normal Form (CNF). To do this we must ensure that the grammar does not contain empty productions, non-terminal unit productions, non-unit terminal productions and non-unit productions of more than 2 elements.

Thus, to maintain structured code, we have decided to make a subclass of the `Gramatica` class seen earlier, which we have called (`Gramatica_FNC`), where we add a `_forma_normal_chomsky` method that will be called when initializing the class

- Elimination of empty productions ($A \to \epsilon$)
- Elimination of unit productions ($A \to B$)
- Substitution of terminals in long productions ($A \to a\ b$)
- Conversion of long productions to binary ($A \to BCD$)

Results Validation

```
S -> NP VP
NP -> Det N, Det Adj N
VP -> V NP
Det -> les
N -> llistes , estructures
Adj -> millors
V -> són

CNF Grammar:
S -> NP VP
NP -> X1 N, Det N
VP -> V NP
Det -> les
N -> estructures , llistes
Adj -> millors
V -> són
X1 -> Det Adj

Sentence: 'les llistes són les millors estructures' -> True
Sentence: 'les llistes són les millors' -> False
```

The transformation to Chomsky Normal Form (CNF) is correct because all productions meet the requirements: they are of the form A → BC or A → a, without mixing terminals and non-terminals. The original production NP → Det Adj N is correctly decomposed through the new variable X1 → Det Adj, and thus NP → X1 N follows CNF.

The sentence "les llistes són les millors estructures" returns True because it can be derived according to the grammar. The subject is generated with NP → Det N → les llistes, the verb with V → són, and the complement with NP → X1 N, where X1 → Det Adj → les millors, and N → estructures. All rules follow CNF and cover all components of the sentence.

In contrast, the sentence "les llistes són les millors" returns False because the production NP → X1 N requires a noun after the adjective. The sequence "les millors" can only generate X1, but does not form an NP without the N. Therefore, there is no complete derivation according to the grammar, and the sentence does not belong to the language.

4.2 extensio2.py: Probabilistic CKY

In this extension we have added the implementation of the probabilistic CKY algorithm, which uses a probabilistic grammar (PCFG). The table contains, instead of sets, tuples with the maximum production probability and the corresponding element. It returns a tuple with a boolean indicating if the word is derivable and the probability of the most probable production of the input string.

It should be noted that since we had to restructure the initial grammar representation to include probabilities in tuples, we could not make a subclass as in extension 1. So we have created a new class (`Gramatica_Probabilistica`) based on the previous ones but making the necessary changes to use the new grammar structure.

Thus, methods like the CNF transformation have had to be redesigned including probability calculations for the new rules. On the other hand, the new methods added to the class are the following:

For the implementation of probabilistic CKY, new methods had to be created and existing ones adapted:

- `algorisme_cky(frase)` → `algorisme_pcky(frase)`: adaptation of the algorithm to include the probabilities of each rule.
- `crear_arbre_gramatical(taula)`: calls the private method `_construir_arbre(taula)` to build the grammatical tree of highest probability recursively.
- `display_arbre()`: calls the private method `_mostrar_arbre(node,depth)` where the node starts being the entire tree and the depth is zero. It traverses the tree in preorder and prints each node taking into account the hierarchical relationship visually.

Results Validation

```
S -> NP VP (1.0)
NP -> Det N (0.7) | Det AN (0.3)
AN -> Adj N (1.0)
VP -> V NP (1.0)
Det -> les (1.0)
N -> llistes (0.5) | estructures (0.5)
Adj -> millors (1.0)
V -> són (1.0)

Sentence: 'les llistes són les millors estructures' -> True, prob: 0.052500

S (p=5.25e-02)
|— NP (p=3.50e-01)
|    |— Det → 'les' (p=1.00e+00)
|    |— N → 'llistes' (p=5.00e-01)
|— VP (p=1.50e-01)
|    |— V → 'són' (p=1.00e+00)
|    |— NP (p=1.50e-01)
|    |    |— Det → 'les' (p=1.00e+00)
|    |    |— AN (p=5.00e-01)
|    |    |    |— Adj → 'millors' (p=1.00e+00)
|    |    |    |— N → 'estructures' (p=5.00e-01)

Sentence: 'les llistes són les millors' -> False, prob: 0.0
```

The sentence "les llistes són les millors estructures" is accepted by the grammar, as shown by the syntactic tree and the total associated probability is 0.0525. The initial symbol S decomposes into NP VP, where NP → Det N generates "les llistes" with probability 0.35, and VP → V NP generates "són les millors estructures" with probability 0.15. Within the second NP, the production NP → Det AN with AN → Adj N is used, which allows the sequence "les millors estructures". All applied rules have probability greater than zero, and therefore, the sentence belongs to the grammar.

In contrast, the sentence "les llistes són les millors" cannot be generated, since a noun is missing after the adjective. The grammar requires that AN → Adj N always include a noun, therefore there is no valid derivation. This is reflected in a total probability of 0 and a False boolean result.

Let's analyze another example to check the effectiveness of the method, so we decide to use the grammar from the 2023 PLH exam, where we can very easily check the generated answer since we have the solutions. Thus, we see:

```
Sentence: 'groucho shot an elephant in his pajamas' -> True, prob: 8.64e-12

S (p=8.64e-12)
|— NP → 'groucho' (p=1.00e-02)
|— VP (p=8.64e-10)
|    |— VP (p=2.88e-05)
|    |    |— VP → 'shot' (p=3.00e-02)
|    |    |— NP (p=2.40e-03)
|    |    |    |— DT → 'an' (p=2.00e-01)
|    |    |    |— NN → 'elephant' (p=3.00e-02)
|    |— PP (p=6.00e-05)
|    |    |— IN → 'in' (p=1.00e-01)
|    |    |— NP (p=6.00e-04)
|    |    |    |— DT → 'his' (p=1.00e-01)
|    |    |    |— NNS → 'pajamas' (p=2.00e-02)
```

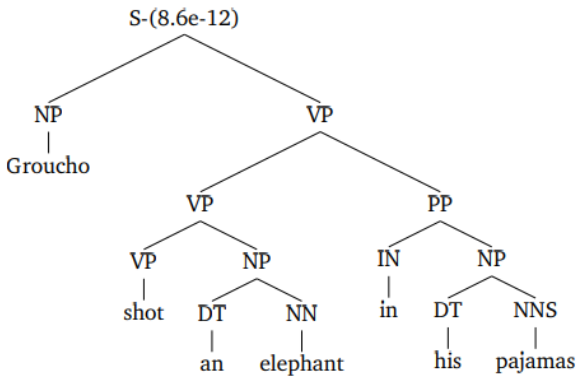

Figure 3: PLH 2023 grammar tree result

In this case, we can visually verify that the generated tree is the same, so the algorithm is working correctly. Furthermore, if we look at the probability of S, in both cases it is 8.6e-12, so the probability of the input strings is also being managed well, confirming the effectiveness of the implemented methods.

5 Learning Assessment

This practice has allowed us to understand and apply the CKY algorithm, a fundamental tool in syntactic analysis of formal languages. Furthermore, implementing extensions such as CNF conversion and the probabilistic version has made us delve deeper into the treatment of formal grammars and the construction of efficient algorithms for computational linguistic analysis.

Additionally, this work has allowed us to review concepts seen both in this course and in others such as human language processing, so we have been able to deepen a concept that is relevant from various areas.

We have also learned to design effective tests and proofs to validate our algorithms, in addition to reviewing basic concepts such as ordered code structures in classes seen in previous courses, so it has allowed us to recover good practices relevant to our future as programmers.

Finally, we have also had to work collaboratively in the development of solid and well-structured software, which has allowed us to learn to manage a project in a structured way by delegating tasks to reach an objective.

6 Other Considerations

- The prohibition on the use of external modules and artificial intelligence tools during the practice has been strictly respected.
- All implementation has been done manually and we understand in depth the functioning of the delivered code.
- A modular code structure has been chosen to facilitate its extension and maintenance.
- The code has been meticulously documented providing relevant information for each method, class and function, adding comments on the most complex code fragments to understand to facilitate reader comprehension.

Annex

Test Results

```
Selecciona una opció:
1. Test de l'algoritme CKY
2. Test de la conversió a FNC(Extensió 1)
3. Test de la conversió a FNC i l'algoritme CKY (Extensió 1 + base)
4. Test de l'algoritme PCKY (Extensió 2)
5. Tots els tests (CKY, FNC i PCKY)
6. Sortir
Introdueix el número de l'opció: 5

Realitzant tots els tests:
Test de l'algoritme CKY:

Prova amb la gramàtica:

S -> a, X A, A X, b
A -> R B
B -> A X, b, a
X -> a
R -> X B

Frase: 'a' -> True
Frase: 'b' -> True
Frase: 'aa' -> False
Frase: 'ab' -> False
Frase: 'ba' -> False
Frase: 'aba' -> False
Frase: 'aaa' -> False
Frase: 'bab' -> False
Frase: 'abab' -> False
Frase: 'aaaabbbb' -> False
Frase: '' -> False

Prova amb la gramàtica:

S -> A B, C D, C B, S S
A -> B C, a
B -> S C, b
C -> D D, b
D -> B A

Frase: 'a' -> False
Frase: 'b' -> False
Frase: 'aa' -> False
Frase: 'ab' -> True
Frase: 'ba' -> False
Frase: 'aba' -> False
Frase: 'aaa' -> False
```

```
Frase: 'bab' -> False
Frase: 'abab' -> True
Frase: 'aaaabbbb' -> True
Frase: '' -> False

Prova amb la gramàtica:

S -> A B,
A -> a,
B -> b

Frase: 'a' -> False
Frase: 'b' -> False
Frase: 'aa' -> False
Frase: 'ab' -> True
Frase: 'ba' -> False
Frase: 'aba' -> False
Frase: 'aaa' -> False
Frase: 'bab' -> False
Frase: 'abab' -> False
Frase: 'aaaabbbb' -> False
Frase: '' -> True

Prova amb la gramàtica:

S -> NP VP
NP -> DT NN, DT NNS, groucho, shot, elephant, NP PP
PP -> IN NP
VP -> VP PP, VP NP, shot
NN -> shot, elephant
NNS -> pajamas
DT -> an, his
IN -> in

Frase: 'his pajamas' -> False
Frase: 'groucho in pajamas' -> False
Frase: 'elephant in pajamas' -> False
Frase: 'groucho shot an elephant in his pajamas' -> True
Frase: '' -> False

Test de la conversió a FNC:

Prova amb la gramàtica en FNC:

S -> A B,
A -> a A, a,
B -> b B, b

Gramatica FNC:
S -> A B
A -> a, T1 A
```

```
B -> b, T2 B
T1 -> a
T2 -> b

Frase: 'a' -> False
Frase: 'b' -> False
Frase: 'aa' -> False
Frase: 'ab' -> True
Frase: 'ba' -> False
Frase: 'aba' -> False
Frase: 'aaa' -> False
Frase: 'bab' -> False
Frase: 'abab' -> False
Frase: 'aaaabbbb' -> True

Prova amb la gramàtica en FNC:

S -> A B C, a
A -> a A, b,
B -> b B, c
C -> c C, d, B

Gramatica FNC:
S -> a, X1 C
A -> T1 A, b
B -> c, T2 B
C -> c, T2 B, d, T3 C
T1 -> a
T2 -> b
T3 -> c
X1 -> A B

Frase: 'a' -> True
Frase: 'b' -> False
Frase: 'aa' -> False
Frase: 'ab' -> False
Frase: 'ba' -> False
Frase: 'aba' -> False
Frase: 'aaa' -> False
Frase: 'bab' -> False
Frase: 'abab' -> False
Frase: 'aaaabbbb' -> False

Prova amb la gramàtica en FNC:

S -> NP VP
NP -> Det N, Det Adj N
VP -> V NP
Det -> les
N -> llistes, estructures
Adj -> millors
```

```
V -> són

Gramatica FNC:
S -> NP VP
NP -> X1 N, Det N
VP -> V NP
Det -> les
N -> estructures, llistes
Adj -> millors
V -> són
X1 -> Det Adj

Frase: 'les llistes són les millors estructures' -> True
Frase: 'les llistes són les millors' -> False
Frase: 'les llistes són estructures' -> False
Frase: 'les estructures són les millors llistes' -> True

Prova amb la gramàtica en FNC:

S -> NP VP
NP -> PRP NN, PRP NNS, NNP, NP AP
VP -> VBD NP, VP AP
AP -> VBG PP
PP -> IN NP
NNP -> john
PRP -> your, his
NN -> brother
NNS -> glasses
IN -> with
VBD -> saw
VBG -> playing

Gramatica FNC:
S -> NP VP
NP -> NP AP, PRP NN, john, PRP NNS
VP -> VBD NP, VP AP
AP -> VBG PP
PP -> IN NP
NNP -> john
PRP -> his, your
NN -> brother
NNS -> glasses
IN -> with
VBD -> saw
VBG -> playing

Frase: 'john saw your brother' -> True
Frase: 'john saw his glasses' -> True
Frase: 'john playing with your brother' -> False
Frase: 'john saw your brother playing with his glasses' -> True
```

```
Test de l'algoritme PCKY:

Prova amb la gramàtica probabilística:

S -> A B (0.6) | B A (0.4)
A -> a (0.7) | A A (0.3)
B -> b (1.0)

Frase: 'a' -> False, prob: 0.0
Frase: 'b' -> False, prob: 0.0
Frase: 'aa' -> False, prob: 0.0
Frase: 'ab' -> True, prob: 0.420000

S (p=4.20e-01)
|- A -> 'a' (p=7.00e-01)
|- B -> 'b' (p=1.00e+00)

Frase: 'ba' -> True, prob: 0.280000

S (p=2.80e-01)
|- B -> 'b' (p=1.00e+00)
|- A -> 'a' (p=7.00e-01)

Frase: 'aba' -> False, prob: 0.0
Frase: 'aaa' -> False, prob: 0.0
Frase: 'bab' -> False, prob: 0.0
Frase: 'abab' -> False, prob: 0.0
Frase: 'aaaabbbb' -> False, prob: 0.0

Prova amb la gramàtica probabilística:

S -> NP VP (1.0)
NP -> Det N (0.7) | Det AN (0.3)
AN -> Adj N (1.0)
VP -> V NP (1.0)
Det -> les (1.0)
N -> llistes (0.5) | estructures (0.5)
Adj -> millors (1.0)
V -> són (1.0)

Frase: 'les llistes són les millors estructures' -> True, prob: 0.052500

S (p=5.25e-02)
|- NP (p=3.50e-01)
|    |- Det -> 'les' (p=1.00e+00)
|    |- N -> 'llistes' (p=5.00e-01)
|- VP (p=1.50e-01)
|    |- V -> 'són' (p=1.00e+00)
|    |- NP (p=1.50e-01)
|    |    |- Det -> 'les' (p=1.00e+00)
|    |    |- AN (p=5.00e-01)
```

```
|   |   |   |-- Adj -> 'millors' (p=1.00e+00)
|   |   |   |-- N -> 'estructures' (p=5.00e-01)

Frase: 'les llistes són les millors' -> False, prob: 0.0
Frase: 'les llistes són estructures' -> False, prob: 0.0
Frase: 'les estructures són les millors llistes' -> True, prob: 0.052500

S (p=5.25e-02)
|-- NP (p=3.50e-01)
|   |-- Det -> 'les' (p=1.00e+00)
|   |-- N -> 'estructures' (p=5.00e-01)
|-- VP (p=1.50e-01)
|   |-- V -> 'són' (p=1.00e+00)
|   |-- NP (p=1.50e-01)
|   |   |-- Det -> 'les' (p=1.00e+00)
|   |   |-- AN (p=5.00e-01)
|   |   |   |-- Adj -> 'millors' (p=1.00e+00)
|   |   |   |-- N -> 'llistes' (p=5.00e-01)


Prova amb la gramàtica probabilística:

S -> NP VP (1.0)
NP -> DT NN (0.4) | DT NNS (0.3) | groucho (0.01) | shot (0.03) | elephant (0.04
PP -> IN NP (1.0)
VP -> VP PP (0.5) | VP NP (0.4) | shot (0.03)
NN -> shot (0.02) | elephant (0.03)
NNS -> pajamas (0.02)
DT -> an (0.2) | his (0.1)
IN -> in (0.1)

Frase: 'his pajamas' -> False, prob: 0.0
Frase: 'groucho in pajamas' -> False, prob: 0.0
Frase: 'elephant in pajamas' -> False, prob: 0.0
Frase: 'groucho shot an elephant in his pajamas' -> True, prob: 0.000000

S (p=8.64e-12)
|-- NP -> 'groucho' (p=1.00e-02)
|-- VP (p=8.64e-10)
|   |-- VP (p=2.88e-05)
|   |   |-- VP -> 'shot' (p=3.00e-02)
|   |   |-- NP (p=2.40e-03)
|   |   |   |-- DT -> 'an' (p=2.00e-01)
|   |   |   |-- NN -> 'elephant' (p=3.00e-02)
|   |-- PP (p=6.00e-05)
|   |   |-- IN -> 'in' (p=1.00e-01)
|   |   |-- NP (p=6.00e-04)
|   |   |   |-- DT -> 'his' (p=1.00e-01)
|   |   |   |-- NNS -> 'pajamas' (p=2.00e-02)
```

```
Prova amb la gramàtica probabilística:

S -> NP VP (1.0)
NP -> PRP NN (0.5) | PRP NNS (0.3) | john (0.1) | NP AP (0.1)
NNP -> john (1.0)
PRP -> your (0.6) | his (0.4)
NN -> brother (1.0)
NNS -> glasses (1.0)
AP -> VBG PP (1.0)
PP -> IN NP (1.0)
IN -> with (1.0)
VBD -> saw (1.0)
VBG -> playing (1.0)
VP -> VBD NP (0.4) | VP AP (0.6)

Frase: 'john saw your brother' -> True, prob: 0.012000

S (p=1.20e-02)
|— NP → 'john' (p=1.00e-01)
|— VP (p=1.20e-01)
|    |— VBD → 'saw' (p=1.00e+00)
|    |— NP (p=3.00e-01)
|    |    |— PRP → 'your' (p=6.00e-01)
|    |    |— NN → 'brother' (p=1.00e+00)

Frase: 'john saw his glasses' -> True, prob: 0.004800

S (p=4.80e-03)
|— NP → 'john' (p=1.00e-01)
|— VP (p=4.80e-02)
|    |— VBD → 'saw' (p=1.00e+00)
|    |— NP (p=1.20e-01)
|    |    |— PRP → 'his' (p=4.00e-01)
|    |    |— NNS → 'glasses' (p=1.00e+00)

Frase: 'john playing with your brother' -> False, prob: 0.0
Frase: 'john saw your brother playing with his glasses' -> True, prob: 0.000864

S (p=8.64e-04)
|— NP → 'john' (p=1.00e-01)
|— VP (p=8.64e-03)
|    |— VP (p=1.20e-01)
|    |    |— VBD → 'saw' (p=1.00e+00)
|    |    |— NP (p=3.00e-01)
|    |    |    |— PRP → 'your' (p=6.00e-01)
|    |    |    |— NN → 'brother' (p=1.00e+00)
|    |— AP (p=1.20e-01)
|    |    |— VBG → 'playing' (p=1.00e+00)
|    |    |— PP (p=1.20e-01)
|    |    |    |— IN → 'with' (p=1.00e+00)
|    |    |    |— NP (p=1.20e-01)
```

```
|   |   |   |   |— PRP → 'his' (p=4.00e-01)
|   |   |   |   |— NNS → 'glasses' (p=1.00e+00)
```