

UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA

Tecnologie del Linguaggio Naturale

Parte 1

Author

Giuseppe BIONDI

13 giugno 2021



Indice

1	Introduzione	2
2	Algoritmo CKY	2
2.1	La struttura dati per la grammatica	2
2.2	La matrice ad oggetti	3
2.3	L'algoritmo	4
3	Lingua Dothraki	6

1 Introduzione

Il seguente progetto è composto in due parti: La prima descriverà l'implementazione dell'algoritmo CKY applicato alle grammatiche L1 e Dothraki. Saran descritte la grammatica, in particolare la struttura dati per memorizzare le regole e i metodi per cercarle, e la matrice ad oggetti implementata per l'algoritmo stesso. Nella seconda parte sarà descritta l'implementazione della grammatica Dothraki con semantica.

Per semplificare la lettura del risultato dell'algoritmo CKY son state implementate due applicazioni tramite la libreria PyQt5 per rispettivamente la matrice di output e gli alberi sintattici risultanti.

2 Algoritmo CKY

2.1 La struttura dati per la grammatica

È stata implementata una classe dedicata per la gestione della grammatica. La classe `grammar.py` fornisce i metodi per la parsificazione da file di testo e per il retrieval delle regole. Per ottimizzare la ricerca della regola da applicare si è deciso di memorizzare in due dictionary separate le regole lessicali e quelle sintattiche. Una dictionary consiste in una collezione di coppie *key - value*. Infine, le regole son salvate invertite: per esempio data una regola del tipo $S \rightarrow NP VP$ andremo a memorizzarle $NP VP \rightarrow S$.

Per implementare tali dictionary è stata scelta la classe `defaultdict`, fornendo come base `set` per evitare di avere due key ripetute. Di conseguenza il *value* della nostra dictionary sarà una lista perchè può capitare che alcune regole condividano lo stesso corpo ma con testa diversa.

\mathcal{L}_1 in CNF
$S \rightarrow NP VP$
$S \rightarrow XI VP$
$XI \rightarrow Aux NP$
$S \rightarrow book \mid include \mid prefer$
$S \rightarrow Verb NP$
$S \rightarrow X2 PP$
$S \rightarrow Verb PP$
$S \rightarrow VP PP$
$NP \rightarrow I \mid she \mid me$
$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det Nominal$
$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal Noun$
$Nominal \rightarrow Nominal PP$
$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb NP$
$VP \rightarrow X2 PP$
$X2 \rightarrow Verb NP$
$VP \rightarrow Verb PP$
$VP \rightarrow VP PP$
$PP \rightarrow Preposition NP$

Figura 1: Grammatica L1 in CNF dal capitolo 13 del libro di Jurafsky

La classe fornisce due metodi per estrarre tutte le regole dati i tag o le parole. Queste regole restituiscono una lista di tag estratti dalle dictionary descritte. Per esempio, nella *grammatica L1* in figura 1, se cerchiamo le regole per 'Verb NP' con il comando `get_rules_for_tag('Verb NP')` otterremo ['S', 'VP', 'X2']

2.2 La matrice ad oggetti

Per l'implementazione della matrice necessaria per l'applicazione dell'algoritmo CKY si è scelto di costruire due classi di appoggio:

- La classe `CkyNode` rappresenta il nodo che servirà poi nell'albero sintattico qual'ora si riuscisse a ottenere il simbolo iniziale della grammatica in uso. Questa classe contiene elementi descrittivi quali: *Il valore del nodo (il tag)*, il riferimento ai figli *left* e *right*, la posizione nella matrice e l'eventuale parola di riferimento se si tratta di un nodo foglia

```
class CkyNode:

    def __init__(self, value, x, y, leftNode=None, rightNode=None, word=None):
        self.value = value
        self.leftNode = leftNode
        self.rightNode = rightNode
        self.x = x
        self.y = y
        self.word = word
```

Figura 2: CkyNode, screen del codice

- Essendo che per ogni posizione della matrice dobbiamo memorizzare più tag (o nodi), è necessario costruire una lista per contenerli, per rendere il codice più leggibile tale struttura è stata implementata in una nuova classe `Cell`.

In questa sono stati inseriti i metodi `add_word(parola_in_input)` e `add_tag(left_cell, right_cell)`: Il primo, data la parola in input interroga la grammatica e per ogni tag associato a tale parola crea un *CkyNode*. Il secondo date due celle, per ogni combinazione di *CkyNode* contenuti in esse, interroga la grammatica per estrarre i tag risultanti dalla combinazione e per ogni tag crea un *CkyNode*.

Come notiamo in figura 3, nelle parti evidenziate, nell'ultimo caso il nodo creato dovrà contenere il riferimento ai *CkyNode left/right* da cui è stato generato, nel primo metodo venivano assegnati di default a *None*, al contrario non conterrà un valore per il campo *word* che interessa solo i nodi foglia.

Dunque avremo che una matrice è composta da una *Cell* ad ogni posizione, questo vale solo per le posizioni utilizzate ovvero quelle sopra la diagonale della matrice, ed ogni *Cell* conterrà un array di *CkyNode*, figura 4.

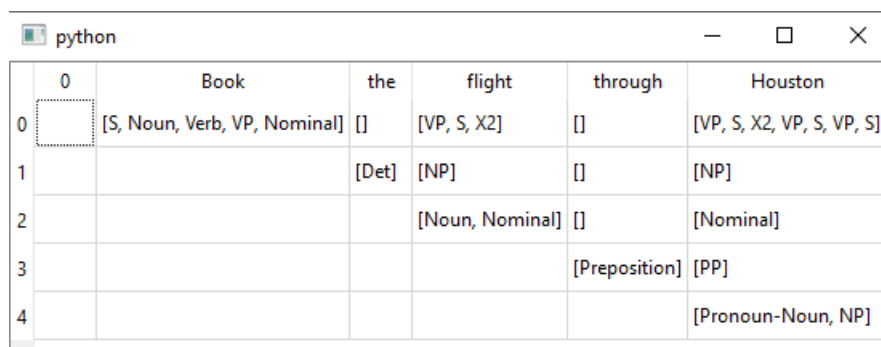
```

def add_word(self, word):
    for tag in self.grammar.get_rules_for_word(word):
        self.node_array.append(CkyNode(tag, self.x, self.y, word=word))

def add_tag(self, cell_left, cell_right):
    for cky_node_left in cell_left.node_array:
        for cky_node_right in cell_right.node_array:
            tag_combination = cky_node_left.value + ' ' + cky_node_right.value
            for result_tag in self.grammar.get_rules_for_tag(tag_combination):
                cky_node = CkyNode(result_tag, self.x, self.y, cky_node_left, cky_node_right)
                self.node_array.append(cky_node)

```

Figura 3: CkyNode, screen del codice



	0	Book	the	flight	through	Houston
0		[S, Noun, Verb, VP, Nominal]	[]	[VP, S, X2]	[]	[VP, S, X2, VP, S, VP, S]
1			[Det]	[NP]	[]	[NP]
2				[Noun, Nominal]	[]	[Nominal]
3					[Preposition]	[PP]
4						[Pronoun-Noun, NP]

Figura 4: Matrice ad oggetti risultante dall'applicazione dell'algoritmo alla frase "Book the flight through Houston"

2.3 L'algoritmo

Per la scrittura dell'algoritmo si è seguito lo pseudocodice fornito nelle slide del corso. Si inizia creando la matrice di partenza usando la libreria *numpy*. Il primo ciclo parte dalla posizione 1 per lasciare la prima colonna vuota e restare il più fedele possibile alle slide del corso. Si crea la *Cell* relativa alla parola di quella colonna e si riempie invocando il metodo `add_word` per creare il nodo foglia.

In seguito si entra nel doppio ciclo per creare le combinazioni delle celle, in questo caso il metodo utilizzato sarà `add_tag` perchè dobbiamo costruire tutte le combinazioni possibili con le celle precedenti, figura 6

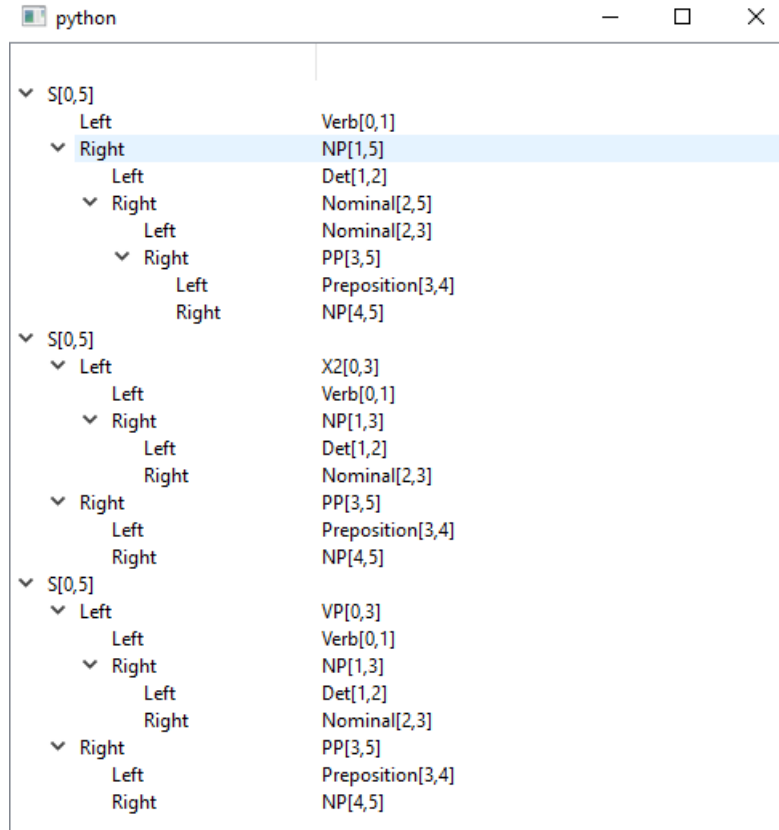


Figura 5: Alberi sintattici risultanti dall'applicazione dell'algoritmo alla frase "Book the flight through Houston"

```
def cky_parse(sentence, gr):
    words = sentence.split()
    table = np.empty((words.__len__(), words.__len__() + 1), dtype=Cell)
    for j in range(1, words.__len__() + 1):
        table[j-1][j] = Cell(gr, j-1, j)
        table[j-1][j].add_word(words[j-1])
        for i in range(j-2, -1, -1):
            for k in range(i+1, j):
                if table[i, j] is None:
                    table[i, j] = Cell(gr, i, j)
                    table[i, j].add_tag(table[i, k], table[k, j])
    return table
```

Figura 6: Algoritmo CKY

3 Lingua Dothraki

Per l'implementazione della grammatica in lingua Dothraki son stati creati due file, uno per l'algoritmo di CKY, che segue la struttura del parser in *grammar.py*, e uno per la semantica, che segue la struttura per *nlk*.

Nella parte lessicale son state classificate le parole seguendo un dizionario online. Per esempio nel caso di *yer/yera* anche se si trattava dello stesso concetto son state considerate come lessici separati.

La grammatica è stata costruita seguendo una guida per lo studio della lingua Dothraki, di conseguenza son state aggiunte regole superflue per le finalità dell'esercizio, come per esempio $S \rightarrow NP \ NP$ in figura 7 e 8

```
S->NP VP | NP Verb | NP NP | Aux Q | Aux VP
Q->NP VP
VP->Verb NP
NP->'Dothraki' | 'Anha' | 'yer' | 'yera' | P NP
P->'ki'
PropN->'Dothraki'
Pronoun->'Anha' | 'yer' | 'yera'
Verb->'zhilak' | 'gavork' | 'astoe'
Aux->'Hash'
```

Figura 7: Grammatica Dothraki per esercizio CKY

Per rappresentare la semantica delle domande si è scelto di emulare la soluzione adottata nel tutorial: <https://www.nltk.org/book/ch10.html>. Si tratta di una soluzione originata per interrogare un database SQL, applicando l'operatore di somma alle semantiche si ottiene un risultato come in figura 9 per la frase *'Hash yer astoe ki Dothraki'*

```

##
% start S
#####
# Grammar Rules
#####
S[SEM = <?vp(?subj)>] -> NP[SEM=?subj] VP[SEM=?vp]
S[SEM = <?v(?subj)>] -> NP[SEM=?subj] V[SEM=?v]
S[SEM = <(is(?subj,?obj))>] -> NP[SEM=?subj] NP[SEM=?obj]
S[SEM = (?aux + <?vp(?np)>)] -> Aux[SEM=?aux] NP[SEM=?np] VP[SEM=?vp]
S[SEM = (?aux + ?vp)] -> Aux[SEM=?aux] VP[SEM=?vp]
NP[SEM=?np] -> PropN[SEM=?np]
NP[SEM=?np] -> Pronoun[SEM=?np]
NP[+by, SEM=?np] -> P[+by] NP[SEM=?np]
VP[SEM=<?v(?obj)>] -> V[SEM=?v] NP[SEM=?obj]
#####
# Lexical Rules
#####
PropN[SEM=<dothraki>] -> 'Dothraki'
P[+by] -> 'ki'
Pronoun[SEM=<i>] -> 'Anha'
Pronoun[SEM=<you>] -> 'yer'
Pronoun[SEM=<you>] -> 'yera'
V[SEM=<\y (\x.love(x,y))>] -> 'zhilak'
V[SEM=<\y (\x.speak(x,y))>] -> 'astoe'
V[SEM=<\x.hungry(x)>] -> 'gavork'
Aux[SEM=<Yes/No>] -> 'Hash'

```

Figura 8: Grammatica Dothraki con semantica per esercizio nltk

```

Label: (Yes/No, speak(you,dothraki))

(S[SEM=(Yes/No, speak(you,dothraki))]
  (Aux[SEM=<Yes/No>] Hash)
  (NP[SEM=<you>] (Pronoun[SEM=<you>] yer))
  (VP[SEM=<\x.speak(x,dothraki)>]
    (V[SEM=<\y x.speak(x,y)>] astoe)
    (NP[SEM=<dothraki>, +by]
      (P[+by] ki)
      (NP[SEM=<dothraki>] (PropN[SEM=<dothraki>] Dothraki))))

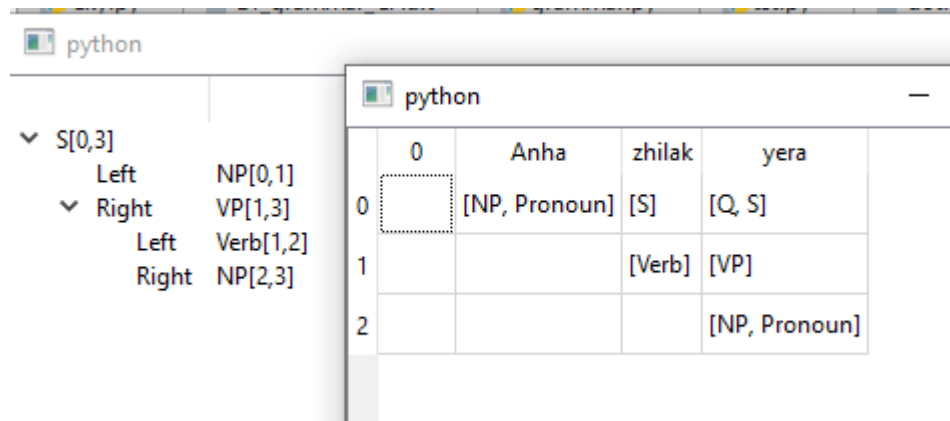
```

Figura 9: Risultato parser semantico nltk per frase: 'Hash yer astoe ki Dothraki'

Risultati

Di seguito saranno mostrati i risultati ottenuti dall'applicazione dell'algoritmo CKY in figura 6 alla grammatica Dothraki in figura 7 rispettivamente per le frasi:

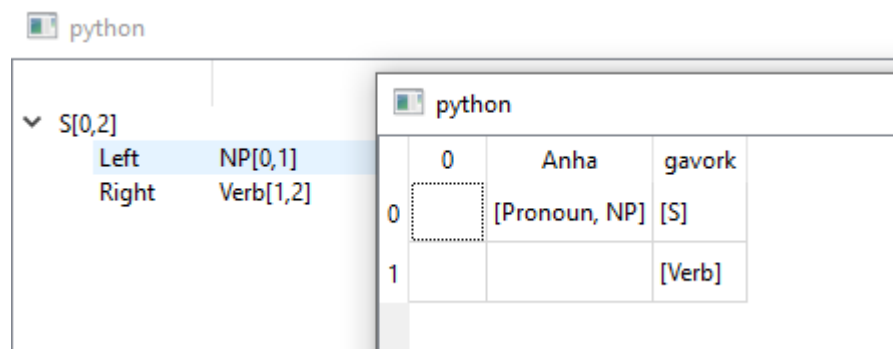
- Anha zhilak yera (*'I love you'*)
- Anha gavork (*'I'm hungry'*)
- Hash yer astoe ki Dothraki (*'Do you speak Dothraki'*)



The screenshot shows a Python IDE with a tree view on the left and a table on the right. The tree view shows a parse tree for the sentence 'Anha zhilak yera' with the root node S[0,3] branching into Left NP[0,1] and Right VP[1,3]. The VP node further branches into Left Verb[1,2] and Right NP[2,3]. The table on the right shows the CKY algorithm results for the sentence 'Anha zhilak yera'.

	0	Anha	zhilak	yera
0	[NP, Pronoun]	[S]	[Q, S]	
1			[Verb]	[VP]
2				[NP, Pronoun]

Figura 10: Risultato algoritmo CKY per frase: 'Anha zhilak yera'



The screenshot shows a Python IDE with a tree view on the left and a table on the right. The tree view shows a parse tree for the sentence 'Anha gavork' with the root node S[0,2] branching into Left NP[0,1] and Right Verb[1,2]. The table on the right shows the CKY algorithm results for the sentence 'Anha gavork'.

	0	Anha	gavork
0	[Pronoun, NP]	[S]	
1			[Verb]

Figura 11: Risultato algoritmo CKY per frase: 'Anha gavork'

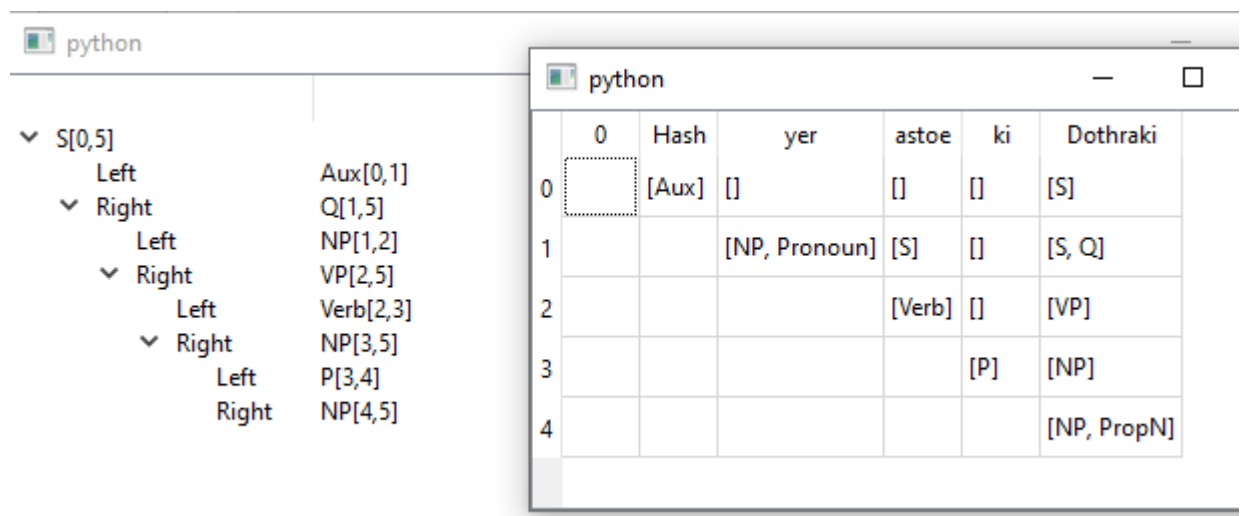


Figura 12: Risultato algoritmo CKY per frase: 'Hash yer astoe ki Dothraki'