Progetto Tecnologie per il Linguaggio Naturale. Consegna CKY: tlhIngan maH!

Mario Scapellato

11 Settembre 2023

1 Introduzione

Il presente elaborato propone l'implementazione from scratch dell'algoritmo CKY, ovvero un algoritmo di parsing impiegato sulle grammatiche libere da contesto, scritte in forma normale di Chomsky (CNF). L'obiettivo è dimostrare il suo utilizzo su delle frasi in lingua Klingon partendo da una grammatica scritta a mano.

Innanzitutto si effettuerà l'implementazione e la verifica dell'algoritmo su delle frasi della grammatica L1 di Jurafsky, successivamente verrà verificato il corretto funzionamento del CKY sulla lingua Klingon, e infine si procederà con un'ulteriore test su frasi di difficile compresione della lingua inglese.

Di seguito verranno presentati i risultati ottenuti e valutati eventuali limiti e/o errori.

2 Sviluppo

In questa sezione si descrivono i principali passaggi che sono stati eseguiti per la costruzione dell'algoritmo CKY, successivamente testato sia sulla grammmatica L1 di Jurafsky sia sulla lingua Klingon:

- Implementazione dell'algoritmo CKY;
- Verifica dell'algoritmo sulla grammatica L1 di Jurafsky per frasi che verranno successivamente identificate;
- Verifica dell'algoritmo sulla lingua Klingon;
- Verifica su due casi particolari della lingua inglese.

2.1 Algoritmo CKY

L'algoritmo CKY può essere inteso come *ricognitore* di un linguaggio, ovvero come un algoritmo in grado di determinare se la grammatica libera da contesto genera o meno una determinata stringa. Nel presente caso, per generare l'algoritmo di *parsing*, si è costruita una struttura dati basandosi su 2 classi:

- *Rule*
- *Cell*

La classe *Rule* definisce le possibili regole di produzione costituite da:

- lato sinistro della regola di produzione
- lato destro della regola di produzione
- risultato della regola di produzione (VP, S, NP ecc..).

La classe *Cell* calcola una regola di produzione all'interno di ogni singola cella. Data una tabella T[i,j], ogni cella si compone di tutti i possibili costituenti che producono la sottostringa dalla posizione i alla posizione j. Pertanto, le celle saranno riempite da sinistra a destra e dal basso verso l'alto. Nella classe *Cell* le funzioni implementate sono le seguenti:

- *add_production()*: si aggiunge una regola di produzione in una determinata cella;
- *get_types()*: si restituisce il risultato della regola di produzione;
- *get_rules()*: si restituisce la lista delle regole di produzione per ogni singola cella.

Tramite queste due classi è stato possibile individuare tutti gli alberi che l'algoritmo CKY genera a partire da un nodo start di una grammatica libera da contesto.

Dopo aver definito la struttura dati rappresentata dalle classi *Cell* e *Rule*, si è sviluppata la terza classe *Grammar*, la quale rappresenta il core dell'algoritmo CKY. Innanzitutto è stato letto il file contenente la grammatica e sono state memorizzate le regole in una struttura a dizionario la cui coppia chiave-valore è rappresentata dalla coppia [*Lista_di_produzione-terminale*]; in seguito è stato sviluppato l'algoritmo CKY, cui parte fondamentale è quella definita dalla funzione *parse* che rappresenta il funzionamento vero e proprio del CKY in veste di *recognizer*.
La parte principale della funzione *parse()* è il triplo ciclo presente al suo interno: si considerano sottostringhe di lunghezza crescente partendo da coppie di token fino ad arrivare all'intera lunghezza della frase. Per ogni sottostringa si determina l'esistenza di una regola in CNF che può derivarlo.

```
def parse(self,sentence):
    self.number_of_trees = 0
    self.tokens = sentence.split()
    self.length = len(self.tokens)
    self.parse_table = [ [Cell() for x in range(self.length - y)] for y in range(self.length) ]

    #Process the first line. In questo ciclo for applico le regole della grammatica ad ogni token della frase.
    for x, t in enumerate(self.tokens):
        r = self.apply_rules(t) #vedo le possibili regole per il token t
        if r == None:
            print("La parola " + str(t) + " non e presente nella grammatica.") #restituisce un errore se il token non è presente nella grammatica
        else:
            for w in r: #per ogni regola di produzione che posso applicare al token t
                self.parse_table[0][x].add_production(w,Rule(t,None,None),None) #aggiungo la regola di produzione alla tabella di parsing nella posizione


    #Run CYK-Parser.
    for l in range(2,self.length+1): #parto con sottostringhe di lunghezza l=2 e arrivo fino alla lunghezza della frase
        for s in range(1,self.length-l+2):
            for p in range(1,l-1+1):

                t1 = self.parse_table[p-1][s-1].get_rules()  #primo token
                t2 = self.parse_table[l-p-1][s+p-1].get_rules() #secondo token

                for a in t1: #per ogni regola di produzione nella cella per il token t1
                    for b in t2: #per ogni regola di produzione nella cella per il token t2
                        r = self.apply_rules(str(a.get_type()) + " " + str(b.get_type())) #applico le regole di produzione per la coppia di token a e b
```

Nell'immagine sovrastante si mostra la rappresentazione del triplo ciclo, in cui l rappresenta le coppie di stringhe che iterativamente vengono considerate, p rappresenta il numero di token e s rappresenta gli spostamenti effettuati.

Dopo aver controllato se una coppia di token può essere prodotta da una regola di produzione o meno, si è restituita la lista completa delle regole applicate su ogni singola frase, così come mostra la seguente immagine:

```
if r is not None: #se la coppia di token a e b può essere prodotta da una regola di produzione
    for w in r: #per ogni regola di produzione che posso applicare alla coppia di token a e b
        print('Regole Applicate : ' + str(w) + '[' + str(l) + ',' + str(s) + ']' +
              ' --> ' + str(a.get_type()) + '[' + str(p) + ',' + str(s) + ']' + ' ' + str(b.get_type())+ '[' + str(l-p) + ',' + str(s+p) + ']')
        self.parse_table[l-1][s-1].add_production(w,a,b)
```

Si è ottenuto pertanto il seguente output:

```
Regole Applicate : X1[2,1] --> Aux[1,1] NP[1,2]
Regole Applicate : S[2,2] --> NP[1,2] VP[1,3]
Regole Applicate : NP[2,4] --> Det[1,4] Nominal[1,5]
Regole Applicate : Nominal[2,5] --> Nominal[1,5] Noun[1,6]
Regole Applicate : S[3,1] --> X1[2,1] VP[1,3]
Regole Applicate : VP[3,3] --> Verb[1,3] NP[2,4]
Regole Applicate : X2[3,3] --> Verb[1,3] NP[2,4]
Regole Applicate : NP[3,4] --> Det[1,4] Nominal[2,5]
Regole Applicate : S[4,2] --> NP[1,2] VP[3,3]
Regole Applicate : VP[4,3] --> Verb[1,3] NP[3,4]
Regole Applicate : X2[4,3] --> Verb[1,3] NP[3,4]
Regole Applicate : S[5,1] --> X1[2,1] VP[3,3]
Regole Applicate : S[5,2] --> NP[1,2] VP[4,3]
Regole Applicate : S[6,1] --> X1[2,1] VP[4,3]
```

2.2 Verifica sulla grammatica L1 di Jurafsky

Come prima verifica dell'algoritmo CKY, è stata considerata la grammatica L1 presente nel libro di Jurafsky:

```
grammar = CFG.fromstring(
    """
S -> NP VP
S -> X1 VP
X1 -> Aux NP
S -> book | include | prefer
S -> Verb NP
S -> X2 PP
S -> Verb PP
S -> VP PP
NP -> I | she | me
NP -> TWA | Houston
NP -> Det Nominal
Nominal -> book | flight | meal | money | morning
Nominal -> Nominal Noun
Nominal -> Nominal PP
VP -> book | include | prefer
VP -> Verb NP
VP -> X2 PP
X2 -> Verb NP
VP -> Verb PP
VP -> VP PP
PP -> Preposition NP

Det -> that | this | the | a
Noun -> book | flight | meal | money | morning
Verb -> book | include | prefer
Pronoun -> I | she | me
Proper-Noun -> Houston | NWA
Aux -> does
Preposition -> from | to | on | near | through
    """
)
```

Le frasi analizzate sono le seguenti:

- *Book the flight through Houston*
- *Does she prefer a morning flight*

Per questa grammatica, cosi come per la grammatica Klingon, si è utilizzata la libreria *nltk* che, data in input una grammatica qualsiasi, permette di trasformare quest'ultima in una CFG. La grammatica presentanta è in *Chomsky Normal Form*, dunque può essere utilizzata per l'esecuzione dell'algoritmo.

Una CFG è in *Chomsky Normal Form* se le produzioni sono nella seguente regola:

- A $\rightarrow$ a
- A $\rightarrow$ aC
- S $\rightarrow \epsilon$

Dopo aver dato in input 2 frasi all'algoritmo di parsing e aver individuato le rispettive regole di derivazione, è stato identificato il numero di possibili alberi associati ai possibili significati della frase stessa.
Di seguito si riporta un esempio di output per la frase *does she prefer a morning flight*:

```
------------ Stringa accettata dal linguaggio ------------
Numero di alberi che si possono generare: 1
----------------------------------------------------------------


------  -----------------  -------------------  -------  -------------------  -------------------
['S']
['S']   ['S']
[]      ['S']              ['S', 'VP', 'X2']
['S']   []                 ['S', 'VP', 'X2']    ['NP']
['X1']  ['S']              []                   ['NP']   ['Nominal']
['Aux']  ['NP', 'Pronoun']  ['S', 'VP', 'Verb']  ['Det']  ['Nominal', 'Noun']  ['Nominal', 'Noun']
does    she                prefer               a        morning              flight
-------  -----------------  -------------------  -------  -------------------  -------------------
```

La procedura di esecuzione dell'algoritmo CKY è di tipo *bottom-up*: si parte dalla sottostringa più piccola per individuare quelle produzioni opportune in ciascuna cella, fino ad arrivare alla sottostringa più grande che rapppresenta la frase intera.

2.3 Verifica sulla lingua Klingon

L'obiettivo principale di questa esercitazione è costruire un parser per la lingua *Klingon*, dunque dopo aver testato il corretto funzionamento dell'algoritmo CKY sulla grammatica L1 di Jurasfky, è stata presa in considerazione una grammatica per la lingua Klingon:

```
klingon = CFG.fromstring(
"""
S -> NP VP
VP -> Verb NP
VP -> Dajatlha | vIlegh | jIHtaH | maH
NP -> Adj Noun
NP -> tlhIngan | Hol | puq | paDaq | jIH
Adj -> tlhIngan
Noun -> tlhIngan | Hol | puq | paDaq | jIH
Verb -> Dajatlha | vIlegh | jIHtaH | maH
"""
)
```

Di seguito sono state riportate le frasi su cui si è verificato il funzionamento dell'algoritmo:

- *tlhIngan Hol Dajatlh'a'?* (Do you speak Klingon?)
- *puq vIlegh jIH* (I see the child)
- *pa'Daq jIHtaH* (I'm in the room)
- *tlhIngan maH!* (We are Klingon!)

Per la costruzione della grammatica Klingon sono state rispettate alcune linee guida presenti sulla pagina di Wikipedia, la quale suggerisce che la lingua Klingon segue un ordine delle parole di tipo *oggetto-verbo-soggetto*, in cui gli avverbi spesso sono posizionati all'inizio della frase, mentre le frasi preposizionali prima dell'oggetto. Tenendo conto di ciò e considerando che la lingua Klingon non presenta articoli, le uniche regole utili a definire la grammatica sono le seguenti:

- S $\rightarrow$ NP VP
- VP $\rightarrow$ Verb NP
- NP $\rightarrow$ Adj Noun

Si noti che la lingua Klingon non possiede dei veri e propri aggettivi come parte distinta del discorso, bensì utilizza alcuni dei propri verbi transitivi.

Dopo aver generato la grammatica, è stato verificato l'utilizzo dell'algoritmo CKY su alcune frasi, ottenendo i seguenti risultati:

```
--------------------  --------------  --------------
['S']
['NP']                ['S']
['NP', 'Adj', 'Noun']  ['NP', 'Noun']  ['VP', 'Verb']
tlhIngan              Hol             Dajatlha
--------------------  --------------  --------------
```

e le seguenti regole:

```
Regole Applicate : NP[2,1] --> Adj[1,1] Noun[1,2]
Regole Applicate : S[2,2] --> NP[1,2] VP[1,3]
Regole Applicate : S[3,1] --> NP[2,1] VP[1,3]
```

2.4 Verifica su un caso particolare

Per concludere si è deciso di verificare ulteriormente l'algoritmo CKY su due frasi di difficile compresione.

La prima frase è *I shot an elephant in my pajamas*, la quale presenta una forte ambiguità in quanto non è chiaro se ad indossare il pigiama è l'elefante o il soggetto *I*. Per poter lavorare sulla frase in questione si è dovuta ampliare ulteriormente la grammatica L1 del Jurafsky, aggiungendo alcune nuove regole:

```
grammar = CFG.fromstring("""
S -> NP VP
S -> X1 VP
X1 -> Aux NP
S -> book | include | prefer
S -> Verb NP
S -> X2 PP
S -> Verb PP
S -> VP PP
NP -> I | she | me
NP -> TWA | Houston
NP -> Det Nominal
NP -> Det Nominal PP
Nominal -> book | flight | meal | money | morning | elephant | pajamas
Nominal -> Nominal Noun
Nominal -> Nominal PP
VP -> book | include | prefer
VP -> Verb NP
VP -> X2 PP
X2 -> Verb NP
VP -> Verb PP
VP -> VP PP
PP -> Preposition NP

Det -> that | this | the | a | an | my
Noun -> book | flight | meal | money | morning
Verb -> book | include | prefer | shot
Pronoun -> I | she | me
Proper-Noun -> Houston | NWA
Aux -> does
Preposition -> from | to | on | near | through | in
""")
```

Attraverso tali regole è stato possibile analizzare la frase tenendo in considerazione entrambe le eventualità: che sia il soggetto *I* ad essere *in my pajamas* o che sia il complemento oggetto *elephant* ad essere *in my pajamas*.

```
----------------  ------------------------------------  -------  -----------  ---------------  -------  -----------
['S', 'S', 'S']
[]                ['S', 'VP', 'X2', 'S', 'VP', 'S', 'VP']
[]                []                                    ['NP']
['S']             []                                    []       ['Nominal']
[]                ['S', 'VP', 'X2']                     []       []           ['PP']
[]                []                                    ['NP']   []           []               ['NP']
['NP', 'Pronoun']  ['Verb']                             ['Det']  ['Nominal']  ['Preposition']  ['Det']  ['Nominal']
I                 shot                                  an       elephant     in               my       pajamas
----------------  ------------------------------------  -------  -----------  ---------------  -------  -----------
```

La seconda frase, sorprendentemente valida per la lingua inglese, è *Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.* Tuttavia, per poterla comprendere risulta necessario tenere in considerazione che:

- Buffalo è un sostantivo plurale che descrive animali noti come *bisonti*;
- Buffalo è una città;
- Buffalo è un verbo che significa *intimidire*;

Pertanto, il significato della frase è *I bisonti della città di Buffalo che sono intimiditi da altri bisonti della città di Buffalo, essi stessi intimidiscono altri bisonti della città di Buffalo.*
Per effettuare questa verifica, tutte le parole della frase sono state scritte in minuscolo, in modo tale che ogni istanza di buffalo potesse appartenere indistintamente a uno dei tre significati sopra elencati. Per lo svolgimento, è stata creata una grammatica minima, ma sufficiente, che consentisse all'algoritmo CKY di trovare gli alberi di parsing per la frase.

```
#Faccio la prova su una grammatica più complessa
grammar2 = CFG.fromstring(
"""
S -> NP VB
S -> NP VP
S -> NN VP
S -> NN VB
NP -> NP RC
NP -> PN NN
RC -> NP VB
VP -> VB NP
VP -> VB NN
VP -> VB PN

NN -> buffalo
PN -> buffalo
VB -> buffalo

"""
)
```

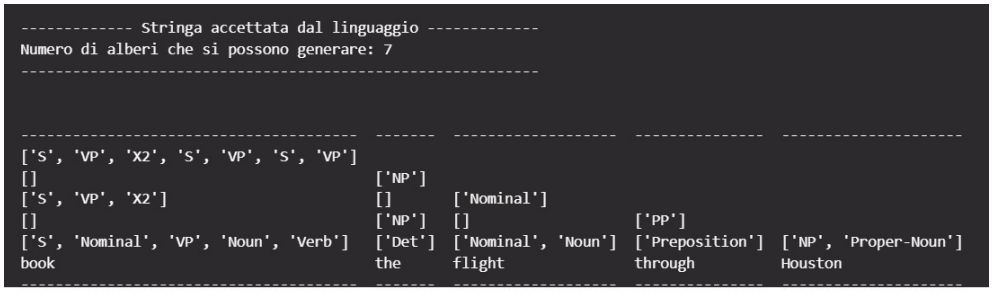
L'output ottenuto è il seguente:

```
------------------------  ------------------------  ------------------------  ------------------------  ------------------------  ------------------------  ----------------------  ----------------
['S', 'NP', 'S', 'NP']
['S', 'S', 'S']           ['S', 'S', 'S']
['VP', 'S', 'RC']         ['VP', 'S', 'RC']         ['VP', 'S', 'RC']
['S', 'NP']               ['S', 'NP']               ['S', 'NP']               ['S', 'NP']
['S', 'S', 'S']           ['S', 'S', 'S']           ['S', 'S', 'S']           ['S', 'S', 'S']           ['S', 'S', 'S']
['S', 'S', 'VP', 'S', 'RC'] ['S', 'S', 'VP', 'S', 'RC'] ['S', 'S', 'VP', 'S', 'RC'] ['S', 'S', 'VP', 'S', 'RC'] ['S', 'S', 'VP', 'S', 'RC'] ['S', 'S', 'VP', 'S', 'RC']
['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']   ['S', 'NP', 'VP', 'VP']
['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']        ['NN', 'PN', 'VB']      ['NN', 'PN', 'VB']
buffalo                   buffalo                   buffalo                   buffalo                   buffalo                   buffalo                   buffalo                 buffalo
------------------------  ------------------------  ------------------------  ------------------------  ------------------------  ------------------------  ----------------------  ----------------
```

L'algoritmo CKY restituisce correttamente una stringa generata dalla grammatica, restituendo 4 possibili alberi sintattici.

3 Considerazioni Finali

L'algoritmo CKY ha restituito risultati ottimali su alcune frasi, mentre su altre ha trovato maggiori difficoltà. Ad esempio, per la frase *book the flight through Houston*, il riconoscitore ha eseguito diverse operazioni utili a capire se la stringa potesse essere accettata o meno, definendo 7 possibili alberi generabili:



Situazioni di questo tipo possono essere dovute all'ambiguità della frase, alle singole parole presenti nella frase stessa, o ancora alla grammatica che si ha a disposizione.
Ad ogni modo, è possibile migliorare tale esecuzione definendo una grammatica *pesata* (PCFG) attraverso cui associare un valore probabilistico a ciascuna regola di produzione. In questo modo può essere restituito l'albero di parsing con peso maggiore date tutte le regole di produzione.