Esercitazione di Tecnologie del Linguaggio Naturale - Parte Prima (prof. Mazzei)

Brunello Matteo (mat. 858867) Caresio Lorenzo (mat. 836021)

Settembre 2023

1A. Algoritmo CKY

Strutture dati e rappresentazione

Per l'implementazione dell'algoritmo abbiamo in primo luogo definito la tipologia di struttura dati sulla quale CKY dovrà operare. Nello specifico, abbiamo definito la tipologia di cella della matrice (triangolare) che utilizzerà CKY: il contenuto di ogni cella deve permettere di tener traccia delle varie regole considerate dall'algoritmo.

Ciò è fondamentale poiché ci permetterà in seguito di fare backtracking e ottenere così le realizzazioni dei vari alberi di parsificazione. Per questa ragione abbiamo definito ogni cella della matrice come una lista di `Entry`. Il tipo `Entry` è un tipo somma che può essere `Terminal` oppure `NonTerminal`. Il primo rappresenta una regola che ha come conseguente un terminale, mentre il secondo rappresenta una regola con due non-terminali come conseguente (queste sono infatti le uniche due tipologie di regole ammesse in una grammatica in *Chomsky Normal Form*, prerequisito di CKY). Questi due non-terminali sono dei puntatori associati a loro volta ad altre `Entry` presenti all'interno della matrice. Come si vedrà in seguito, queste associazioni verranno poi costruite durante l'esecuzione dell'algoritmo.

Implementazione

Dopo aver definito la tipologia di struttura dati, è utile inoltre introdurre alcune funzioni di supporto che implementano alcune operazioni ad alto livello che compaiono nella definizione dell'algoritmo. Più nel dettaglio, definiamo due funzioni rispettivamente per:

1. Trovare nella grammatica tutte le regole consistenti con il terminale dato (`find_consistent_terminal`)
2. Trovare nella grammatica tutte le regole consistenti con due non-terminali dati (`find_consistent_nonterminal`)

```
def find_consistent_terminal(grammar: nltk.grammar.CFG, terminal: str):
  return map(lambda x: Terminal(x.lhs(), terminal), grammar.productions(rhs = terminal))

def find_consistent_nonterminal(grammar: nltk.grammar.CFG, b: NonTerminal, c: NonTerminal):
  result = list()

  for production in grammar.productions():
    if(len(production.rhs()) == 2):
      (b_i, c_i) = production.rhs()
      # Check for all possible entries in both sets
      for entry_b in b:
        for entry_c in c:
          if b_i == entry_b.get_value() and c_i == entry_c.get_value():
            result.append(NonTerminal(production.lhs(), entry_b, entry_c))
```

```
  return result
```

L'implementazione dell'algoritmo diventa a questo punto molto semplice, poiché segue direttamente dalla definizione in pseudocodice data dal libro. Si noti che nell'implementazione seguente gli indici sono stati rivisti poiché nello pseudocodice la lista delle parole inizia con l'indice 1, mentre in Python con l'inidice 0.

```
def cky_parse(words: List[str], grammar: nltk.grammar.CFG) -> List[List[Entry]]:

  table = [[[] for i in range(len(words))] for j in range(len(words))]

  for j in range(0, len(words)):
    table[j][j].extend(find_consistent_terminal(grammar, words[j]))
    for i in range(j, -1, -1):
      for k in range(i, j):
        table[i][j] += find_consistent_nonterminal(grammar, table[i][k], table[k+1][j])

  return table
```

Per definizione, l'algoritmo ritorna una tabella che rappresenta tutte le possibili derivazioni degli alberi di parsificazione. Si avranno tanti alberi tante quante saranno le produzioni con antecedente S (*start symbol*) presenti nella cella alla posizione $[0, len(words)]$. Naturalmente, se tale produzione non compare in quella posizione, la grammatica non copre la frase. Possiamo sfruttare questo fatto per verificare se una frase sia coperta o meno dalla grammatica.

```
def is_covered(words, grammar):
  table = cky_parse(words, grammar)
  for entry in table[0][len(words) - 1]:
    if entry.is_start():
      return True
  return False
```

1B. Grammatica L1

Riportiamo di seguito la grammatica L1 come fornita dal Jurafsky & Martin (J&M). Nella nostra implementazione abbiamo deciso di sfruttare la classe `CFG` del modulo `nltk.grammar`, così da poter sfruttare una rappresentazione comune e ben testata. Per il funzionamento dell'algoritmo CKY è necessario che la grammatica sia in CNF, e così viene riportata di seguito:

- 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*
- Aux → *does*
- Preposition → *from* | *to* | *on* | *near* | *through*

Per coprire tutte le frasi assegnate (si veda la sezione successiva) si è dovuta espandere la grammatica fornita dal J&M, nello specifico per la frase *does she prefer a morning flight*, dove la costruzione *morning flight* non veniva coperta dalla grammatica originale.

Copertura

Una volta definito l'algoritmo di parsing e la grammatica è possibile verificare la copertura delle frasi date, qui riportate:

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

Alle due fornite ne è stata aggiunta una terza, volutamente non coperta dalla grammatica definita sopra. È possibile verificare la copertura di queste frasi con la funzione da noi implementata `is_covered`, che ricordiamo si basa anch'essa sull'utilizzo di CKY.

```
[COVERED]:   "does she prefer a morning flight"

[COVERED]:   "book the flight through Houston"

[NOT COVERED]:   "book the flight through Houston to book"
```

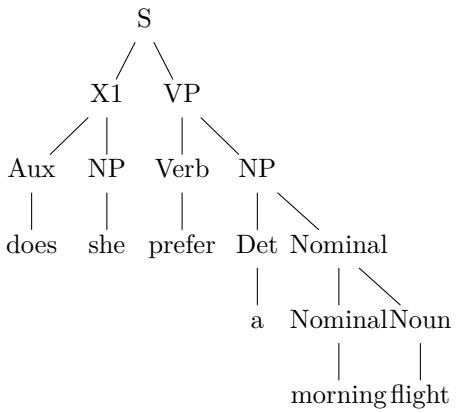
Alberi di parsificazione

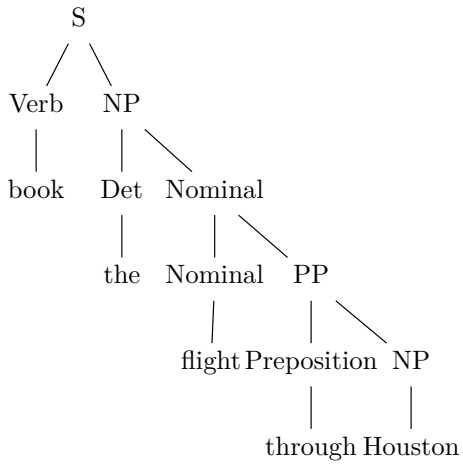
Precedentemente è stato discusso di come l'algoritmo CKY ritorni in forma tabellare tutti i possibili alberi di parsificazione. Grazie all'implementazione di `Entry` è possibile estrarre tutti gli alberi validi seguendo il percorso dei vari *backpointers* con una visita in profondità, partendo dai non-terminali S presenti nella cella a coordinate $[0, len(words)]$.

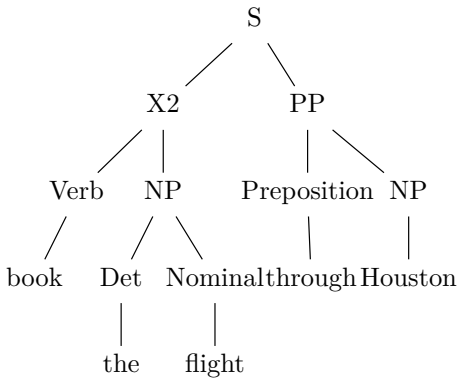
```
def get_parsing_trees(words, grammar):

  parsing_trees = []
  table = cky_parse(words, grammar)

  for entry in table[0][len(words) - 1]:
    # Found an 'S' entry
    if entry.is_start():
      parsing_tree = []
      evaluation_stack = [entry]
      # Find the tree
      while evaluation_stack:
        current_node = evaluation_stack.pop(-1)
```

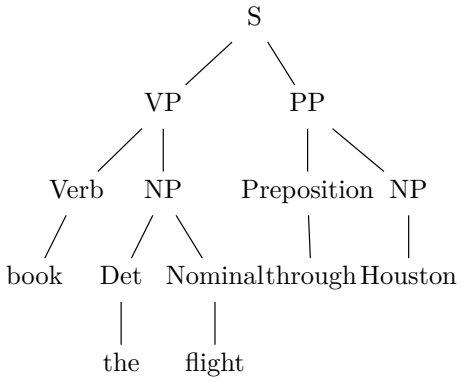
```
        match current_node:
          case NonTerminal(value = non_terminal, left = left_ptr, right = right_ptr):
            parsing_tree.append(current_node)
            evaluation_stack.append(left_ptr)
            evaluation_stack.append(right_ptr)
          case Terminal():
            parsing_tree.append(current_node)
      # Add the tree
      parsing_trees.append(parsing_tree)

  return parsing_trees
```

Di seguito gli alberi di parsificazione ottenuti per le due frasi date (essendo la terza frase non coperta dalla grammatica, non ha alberi di parsificazione corrispondenti).









Si noti come per frasi sintatticamente ambigue, gli alberi di parsificazione sono molteplici.

1C. Grammatica Klingon

Proponiamo ora una grammatica per la lingua Klingon. La metodologia seguita è stata quella di partire creando inizialmente le regole terminali, per poi successivamente creare le regole di composizione grammaticale secondo il Klingon Dictionary.

Per questa ragione, la grammatica ottenuta è molto ristretta, poiché copre solamente le frasi fornite. Secondo una prima analisi, è stata quindi ottenuta la seguente grammatica formale (non in CNF):

- $S \rightarrow$ VP NP
- $S \rightarrow$ VP Pronoun
- $S \rightarrow$ NP VP
- $S \rightarrow$ NP Pronoun
- $VP \rightarrow$ NP VP
- $VP \rightarrow Dajatlh'a' \mid vIlegh \mid jIHtaH$
- $NP \rightarrow$ Noun
- $NP \rightarrow$ Noun Noun
- $NP \rightarrow$ Pronoun
- $Noun \rightarrow pa'Daq \mid puq \mid tlhIngan \mid Hol$
- $Pronoun \rightarrow jIH \mid maH$

Rispetto all'Inglese, il Klingon è una lingua con *word-order* più frequente oggetto-verbo-soggetto, da questo la presenza di regole di riscrittura con VP come primo non-terminale (rispetto a quelle della

grammatica L1 Inglese, dove si ha una maggior frequenza di NP come primo elemento).

Altro tratto caratteristico del Klingon è la mancanza del verbo *essere* (cfr. Klingon Dictionary, §6.3), dove però i pronomi possono essere usati come verbi. Così la frase *tlhIngan maH* (corrispondente all'Inglese *We are Klingon*) è composta semplicemente da un sostantivo e da un pronome. Sempre da questo esempio è possibile notare un'altra caratteristica tipica del Klingon: la mancanza di aggettivi.

Siccome CKY necessita però della grammatica in CNF, la riporteremo in questa forma seguendo la metodologia di trasformazione proposta dal libro.

- S $\rightarrow$ VP NP
- S $\rightarrow$ VP Pronoun
- S $\rightarrow$ NP VP1
- S $\rightarrow$ NP Pronoun
- VP $\rightarrow$ NP VP1
- NP $\rightarrow$ $pa'Daq \mid puq \mid tlhIngan \mid Hol \mid jIH \mid maH$
- NP $\rightarrow$ Noun Noun
- Noun $\rightarrow$ $puq \mid tlhIngan \mid Hol$
- Pronoun $\rightarrow$ $jIH \mid maH$
- VP1 $\rightarrow$ $Dajatlh'a' \mid vIlegh|jIHtaH$

Il terminale VP1 corrisponde ai verbi, si è scelta però questa denominazione rispetto a quella di *Verb* o *V* in quanto i verbi denotati da VP1 non sono semplici verbi ma contengono inoltre particelle e suffissi tipiche del Klingon.

Copertura

Di seguito le quattro frasi su cui si è basato lo sviluppo della grammatica visto sopra:

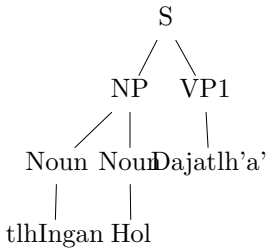
- *tlhIngan Hol Dajatlh'a'* (*Do you speak Klingon?*)
- *puq vIlegh jIH* (I see the child)
- *pa'Daq jIHtaH* (*I'm in the room*)
- *tlhIngan maH* (*We are Klingon!*)

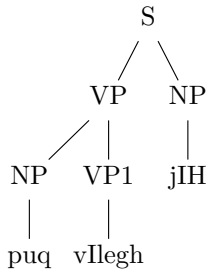
Così come in precedenza, è possibile utilizzare la funzione `is_covered` da noi implementata per verificare la copertura su queste quattro frasi in Klingon.

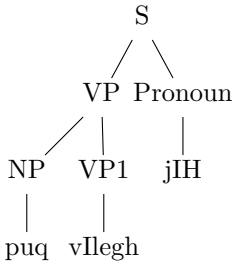
```
[COVERED]:   "tlhIngan Hol Dajatlh'a'"

[COVERED]:   "puq vIlegh jIH"

[COVERED]:   "pa'Daq jIHtaH"

[COVERED]:   "tlhIngan maH"
```

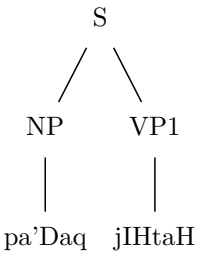
Alberi di parsificazione

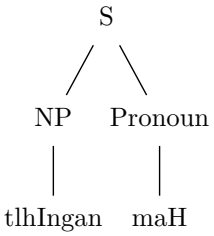
Di seguito gli alberi di parsificazione ottenuti per le quattro frasi date.


S
NP
VP1
Noun
Noun
Dajatlh'a'
tlhIngan
Hol


S
VP
NP
NP
VP1
jIH
puq
vIlegh


S
VP
Pronoun
NP
VP1
jIH
puq
vIlegh


S
NP
VP1
pa'Daq
jIHtaH


S
NP
Pronoun
tlhIngan
maH