Traduttore Italiano - Italiano Yoda

Francesco Odierna, Michele Puca

1 Introduzione

Yoda è un personaggio immaginario della saga fantascientifica di Guerre Stellari. La lingua da lui parlata è lo Yoda, appunto, una lingua fittizia che somiglia all'italiano, se non per l'ordine delle parole.

La sintassi Yoda somiglia alla sintassi *OSV* (Oggetto-Soggetto-Verbo) [3]. In particolare, la sintassi Yoda viene classificata come *XSV*, dove la X indica qualsiasi complemento che si accorda con il verbo indipendentemente dal fatto che sia un oggetto o meno; S indica il soggetto e V il verbo.

Questo tipo di sintassi è molto raro, infatti sono state trovate circa 8-10 lingue in cui si utilizza questa struttura sintattica. Un esempio di lingua XSV è lo jatvingico (o sudoviano), parlato in Galindia e in Sudovia, diventate più tardi Prussia orientale e Lituania sudoccidentale, rispettivamente.

Di seguito sarà presentato un traduttore che permette di passare da una sintassi SVX ad una sintassi XSV, tipica della lingua Yoda.

Nel Capitolo 2 ci sarà un'introduzione alle grammatiche libere dal contesto, dette anche *context free grammars*.

Nel Capitolo 3 verrà presentato l'algoritmo CKY, utilizzato per la parsificazione delle frasi.

Nel Capitolo 4 discuteremo l'implementazione l'algoritmo di traduzione che permette di passare dall'italiano all'italiano Yoda.

Nel Capitolo 5 vedremo le traduzioni prodotte dall'algoritmo e infine ci saranno le conclusioni.

2 Context Free Grammar

In questa sezione introduciamo il lettore alle context free grammar (CFG) [2], in quanto la grammatica utilizzata nell'implementazione dell'algoritmo di parsing è proprio una CFG.

Una CFG genera un linguaggio libero dal contesto. Le regole di produzione di una CFG consistono di una *testa* e di un *corpo* e sono della forma

$S- > aSb$. Nella testa è presente un simbolo non terminale, mentre nel corpo abbiamo un simbolo terminale sia a destra (b) che a sinistra (a) del simbolo non terminale (S). Le CFG nella gerarchia di Chomsky sono dette di Tipo 2. Le regole di produzione sono delle sostituzioni e la struttura fondamentale è la stringa. Il tempo di parsing di una linguaggio generato da un CFG utilizzando algoritmi di programmazione dinamica è pari ad $O(n^3)$.

Le CFG sono molto interessanti perché riconoscibili da un automa a pila non deterministico.

3 Algoritmo CKY

L'algoritmo CKY (Coke Kasami Younger) [1] è un algoritmo di parsing per grammatiche libere dal contesto. L'algoritmo è data-directed, quindi bottom-up, left-to-right e utilizza la programmazione dinamica, ossia memorizza le soluzioni parziali in modo da non dover ripetere calcoli già effettuati.

CKY calcola tutti i possibili parse in tempo $O(n^3)$, che coincide anche con il caso medio. La grammatica in input deve essere una CFG in forma normale di Chomsky (CNF) [2], ossia le regole della grammatica devono avere nel corpo esattamente due simboli non terminali o un simbolo terminale: $A \rightarrow BC$ o $A \rightarrow a$. Qualsiasi grammatica può essere trasformata in una grammatica equivalente in CNF.

L'algoritmo utilizza una matrice $N \times N$, dove N è il numero di parole che costituiscono la frase, per memorizzare i risultati parziali della computazione così da aumentare l'efficienza del parsing.

Visto che il parsing avviene bottom-up, l'algoritmo parte dalle parole e applica tutte le possibili regole per popolare la matrice. Se la parsificazione ha successo per una frase di dimensione N, nella cella $[0, N]$ sarà presente il costituente S, che produce la frase in input.

Ogni cella della matrice può contenere più costituenti, in quanto la matrice mantiene tutte possibili derivazioni, ossia i diversi modi di generare una stessa parola o una serie di parole. Per tale motivo è necessario il *linking* fra i costituenti in modo da restituire una soluzione unica.

A causa dell'ambiguità del linguaggio potremmo avere più di un costituente S nella cella $[0, N]$, in questo caso è necessaria un'analisi semantica per disambiguare. Nella nostra implementazione il problema dell'ambiguità non è stato trattato. Viene restituito sempre il primo costituente S presente nella cella $[0, N]$.

In Figura 1 troviamo lo pseudo-codice dell'algoritmo.

```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
    for all {A | A → words[j] ∈ grammar}
          table[j − 1, j] ← table[j − 1, j] ∪ A
    for i ← from j − 2 downto 0 do
      for k ← i + 1 to j − 1 do
        for all {A | A → BC ∈ grammar and B ∈ table[i,k] and C ∈ table[k, j]}
              table[i,j] ← table[i,j] ∪ A
```

Figura 1: Pseudocodice dell'algoritmo CKY

4 Implementazione

Di seguito verranno descritte le scelte progettuali e l'implementazione dell'algoritmo di traduzione. Per implementare l'algoritmo di traduzione è necessaria una *grammatica*, un *algoritmo di parsing* per ottenere la struttura sintattica della frase da tradurre e un *algoritmo di traduzione* per manipolare l'albero sintattico prodotto dall'algoritmo di parsing.

4.1 Grammatica

La grammatica utilizzata è una CFG in CNF. Sono state modellate le regole grammaticali:

- sintagma nominale (NP)
- sintagma verbale (VP)
- sintagma preposizionale (PP)
- frase (S)

e le seguenti unità sintattiche:

- pronomi personali (PRON)
- nomi (N)
- verbi (V)
- aggettivi (ADJ)
- avverbi (ADV)
- preposizioni (PP)
- articoli (DET)

4.2 Algoritmo di parsing: CKY

L'algoritmo di parsing adottato è CKY, descritto nel Capitolo 3. L'implementazione dell'algoritmo è stata realizza in Python con l'ausilio di alcune librerie di NLTK, una piattaforma che fornisce classi e metodi ad hoc per l'elaborazione e la manipolazione del linguaggio naturale. Il codice è organizzato in due moduli: *utilities* e *main*. Il modulo *main* ha la funzione di caricare la grammatica, invocare l'algoritmo sulle frasi in input ottenendo l'albero sintattico corrispondente e passare quest'ultimo all'algoritmo di traduzione, che effettuerà manipolazione dell'albero. Nel modulo *utilities* troviamo l'implementazione dell'algoritmo CKY e tutte le funzioni di supporto necessarie all'algoritmo CKY e all'algoritmo di traduzione che sarà discusso in seguito.

In Figura 2 troviamo il codice dell'algoritmo.

```
def cky(words, grammar):
    dim = len(words)+1
    # create dim x dim matrix. Each element of the matrix is a list (of tree)
    table = [[[] for i in range(dim)] for j in range(dim)]
    for j in range(1, dim):
        # get le lhs of a rule that has a terminal symbol (word[hj-1]) in the rhs
        lhs = get_lhs_lexical_rule(words[j-1], grammar)
        if(lhs is not None):
            #A -> words[j] and create a Tree
            table[j-1][j].append(Tree(lhs, [words[j-1]]))
        for i in reversed(range(0, j - 1)):
            for k in range(i + 1, j):
                if(len(table[i][k])!=0 and len(table[k][j])!=0):
                    #We look for a rule A -> BC in the grammar
                    #such that B is member of table[i][k] and C is member of table[k][j]
                    A, B, C = get_grammar_rule(table[i][k], table[k][j], grammar)
                    if A is not None:
                        table[i][j].append(Tree(A, [B, C]))
    #if S is table[0][len(words)] then we have a tree for the input sentence
    if len(table[0][dim - 1]) != 0:
        return get_final_tree(table[0][dim - 1])
    else:
       print('Cannot derive the sentence')
```

Figura 2: Implementazione in Python dell'algoritmo CKY

4.3 Algoritmo di traduzione

La procedura di traduzione permette di passare da un albero sintattico SVX ad un albero XSV. L'idea alla base dell'algoritmo è quella di individuare il verbo (V) nell'albero sintattico e a partire dal verbo si individuano il soggetto (S), che si troverà a livello superiore e tutto il resto (X), ossia aggettivi, preposizioni, complemento oggetto etc. Una volta individuati gli elementi SVX questi vengono utilizzati per creare un nuovo albero sintattico della forma XSV. In Figura 3 il codice dell'algoritmo di traduzione.

```
# given a tree in SVX form, it returns a tree in XSV form
def translate_it_yo(tree):
    SUBJ = [Nonterminal("PRON"), Nonterminal("NP"), Nonterminal("N")]
    VERB = [Nonterminal("VP")]
    yoda_tree = Tree("Yoda", [])
    for i in range(len(tree)):
        if(tree[i].label() in SUBJ):
            yoda_tree.insert(1,tree[i])
        if(tree[i].label() in VERB):
            V = tree[i][0]
            X = tree[i][1]
            yoda_tree.insert(0, X)
            yoda_tree.insert(2, V)
    return yoda_tree
```

Figura 3: Codice dell'algoritmo di traduzione

5 Risultati e conclusioni

In questa sezione vediamo il comportamento dell'algoritmo su diverse frasi di esempio. La grammatica utilizzata è in Figura 4.

Le frasi modellate dalla grammatica sono le seguenti:

- Tu avrai novecento anni di età
- Tu hai amici lì
- Noi siamo illuminati
- Il lato oscuro è arduo da vedere
- Tu hai molto da apprendere ancora

```
#Grammar rules
S   ->  N VP | PRON VP | NP VP
NP  ->  PRON VP | N PP | ADJ NP | N ADV | DET NP | N ADJ | ADJ PP | ADV PP | PRON N
VP  ->  V ADJ |  V NP | V ADV
PP  ->  PP N | PP V | PP VP | PP ADJ

#Lexical rules
N       ->  'anni' | 'età' | 'amici' | 'lato' | 'skywalker'| 'futuro' | 'ragazzo' | 'apprendista'
V       ->  'avrai' | 'hai' | 'siamo' | 'è' |'vedere' | 'apprendere' |'corre' | 'sarà'
ADJ     ->  'illuminati' | 'oscuro' | 'arduo' | 'nebuloso' | 'questo' | 'novecento'
PRON    ->  'tu' | 'noi'| 'tuo'
DET     ->  'il'
ADV     ->  'molto' | 'ancora' | 'veloce' | 'lì'
PP      ->  'di' | 'da'
```

Figura 4: CFG in CNF

- Skywalker corre veloce
- Il futuro di questo ragazzo è nebuloso
- Skywalker sarà tuo apprendista

Le traduzioni che ci aspettiamo di ottenere sono le seguenti:

- Novecento anni di età tu avrai
- Amici lì tu hai
- Illuminati noi siamo
- Molto da apprendere ancora tu hai
- Veloce Skywalker corre
- Nebuloso il futuro di questo ragazzo è
- Tuò apprendistà Skywalker sarà

In Figura 5 abbiamo l'albero sintattico di una frase di test e in Figura 6 l'albero sintattico della frase tradotta in Yoda.

A seguire altri esempi di alberi sintattici relativi alle frasi in input e alle frasi tradotte in Yoda.

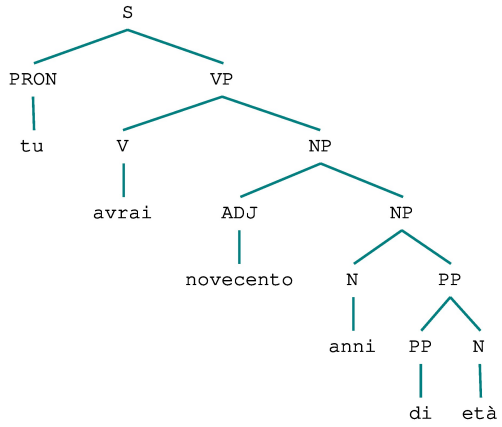

Figura 5: Albero sintattico IT

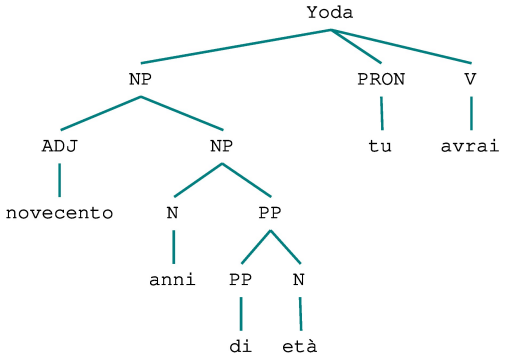

Figura 6: Albero sintattico IT-YO

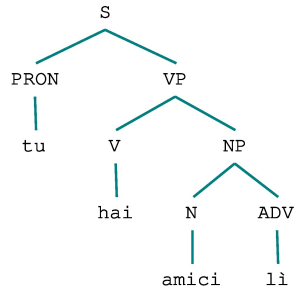

Figura 7: Albero sintattico IT

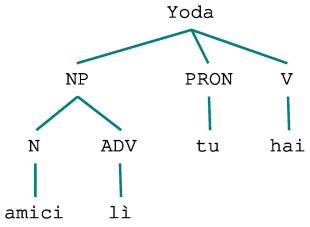

Figura 8: Albero sintattico IT-YO

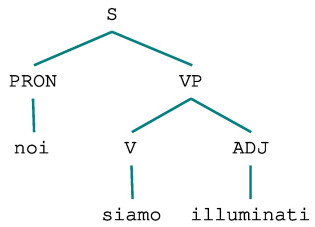

Figura 9: Albero sintattico IT

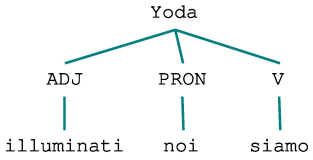

Figura 10: Albero sintattico IT-YO

5.1 Conclusioni

Nel presente lavoro è stato implementato un semplice traduttore Italiano - Italiano Yoda. La traduzione è ottenuta modificando gli alberi sintattici prodotti l'algoritmo CKY con l'ausilio di una CFG. Possiamo dire che l'algoritmo di parsificazione e quello di traduzione si comportano molto bene e la grammatica riesce a catturare tutte le relazioni presenti nelle frasi, infatti per le frasi di esempio si raggiunge un'accuratezza pari al 100%.

Riferimenti bibliografici

[1] Tadao Kasami. «An efficient recognition and syntax-analysis algorithm for context-free languages». In: *Coordinated Science Laboratory Report no. R-257* (1966).

[2] James H Martin e Daniel Jurafsky. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009.

[3] Wikipedia contributors. *Object–subject–verb*. 2019. URL: `https://en.wikipedia.org/w/index.php?title=Object%E2%80%93subject%E2%80%93verb&oldid=911332187`.