Daniele Perlo a.a 2013/14

Relazione Sistemi Cognitivi
Esercitazioni Prof. Mazzei

Punto 1: Scrivere a una CFG per l'italiano (G1) che parsifichi (almeno) le frasi di esempio

La stesura della grammatica Context Free è stata realizzata sulla falsa riga degli esempi visti in classe, come “Paolo ama Francesca”.
La grammatica definisce il livello sintattico su cui possono essere parsificate le frasi d'esempio, utilizzando una struttura a costituenti.
Partendo dalla regola per la sentence S, simbolo iniziale, sono state prodotte le regole per definire i verbi (VP) e le regole per definire i costituenti contenenti i nomi (NP)
La grammatica è in forma normale di Chomsky, in cui sono permesse unicamente le regole di riscrittura del tipo $A \rightarrow B\ C$ e $A \rightarrow t$ con A, B, C simboli non terminali e t simbolo terminale.

Punto 2: Implementare in Java l'algoritmo CKY e provare la grammatica G1 su questo programma per le frasi di esempio

Per la realizzazione dell'algoritmo CKY in Java è stata creata una struttura CFGrammar che contenesse l'alfabeto, la lista dei simboli non terminali e il simbolo di partenza. Per ogni simbolo all'interno di CFGrammar, si tiene traccia dei simboli da cui può essere prodotto e, nel caso si tratti di non terminali, della lista di produzioni che esso genera.
Questa struttura viene caricata dinamicamente prendendo in input la grammatica generata al punto precedente.

L'algoritmo CKY si sviluppa iterativamente (dynamic programming) utilizzando una struttura matriciale contenente liste di simboli estratti dalle parti sinistre delle regole di produzione che corrispondono ai simboli associati alle parole. Si tratta quindi di un parsificatore bottom-up, basato su regole.
Di seguito è riportato un esempio di output di verifica delle frasi sulla grammatica G1.

```
[NP] [CG] []   [NP] []   []   [S]  []   [S]
     [CC] []   []   []   []   []   []   []
          [DT] [NP] []   []   [S]  []   [S]
               [N]  []   []   []   []   []
                    [V]  []   [VP] []   [VP]
                         [DT] [NP] []   []
                              [N]  []   []
                                   [PR] [CT]
                                        [NP]
Dante e una donna lasciano un dono a Virgilio -> true
```

Punto 3: Dotare di semantica la grammatica G1 (G2) e provarla in NLTK

La grammatica G2 è una estensione della grammatica G1 con l'aggiunta di informazioni semantiche ai terminali e ai non terminali, descritte tramite lambda-expressions. Le lambda-expressions sono poi applicate partendo dalle definizioni dei simboli non terminali nell'ordine descritto dalla struttura lessicale a costituenti, in modo tale da produrre, risalendo la struttura ad albero, una semantica per l'intera frase in first order logic.
Le descrizioni semantiche sono state derivate dagli esempi visti a lezione.
L'insieme di librerie NLTK in Python per l'analisi del linguaggio naturale permettono il caricamento della grammatica e la parsificazioni delle frasi di esempio generando le rispettive semantiche in first order logic.
Questo punto è stato ribattezzato TextPlanning per il task di traduzione delle frasi d'esempio dall'italiano all'inglese.
Esempio output:

```
una donna e Dante lasciano un dono a Virgilio →
(exists x.(woman(x) & exists z7.(gift(z7) & give(x,z7,Virgilio)))
& exists z7.(gift(z7) & give(Dante,z7,Virgilio)))
```

Punto 4: Costruire un sentence planner che per ogni formula prodotta dalla grammatica G2 produca un sentence plan (abstract-syntax tree a dipendenze) valido.

Il Sentence Planner è stato realizzato parsificando l'output del Text Plan, ovvero parsificando la semantica delle frasi descritta in first order logic.
Si è utilizzato un approccio basato su espressioni regolari per identificare il predicato che descrive il verbo della frase. In seguito vengono quindi estratti soggetti e, eventualmente, complementi oggetti e complementi di termine.

Per le variabili libere quantificate da “per ogni” e “esiste” sono stati analizzati i predicati per cui vengono applicate, risalendo così alle definizioni, ad esempio, dei nomi comuni che le descrivono.
Per poter inserire nel la pluralità dei nomi così ricavati all'interno del Sentence Plan, sono stati analizzati i quantificatori, in particolare il “per ogni” che descrive la parola “tutti/tutte”.

Di seguito riportato un esempio di SentencePlan in formato JSON.

```
{"OBJ": [{"QUANT": "exist", "NUM": "sg", "NOUN": "gift"}],
"VERB": "give",
"SUBJ": [{"QUANT": "exist", "NUM": "sg", "NOUN": "woman"},
{"NUM": "sg", "NOUN": "Dante"}],
"CTERM": {"NUM": "sg", "NOUN": "Virgilio"}}
```

Punto 5: Usando la libreria SimpleNLG (eventualmente come server attraverso socket o via pipe) implementare un semplice realizer che trasformi i sentence plans in frasi inglesi.

L'implementazione della fase di Linguistic Realization è stata realizzata tramite la libreria SimpleNLG, che permette la costruzioni di frasi dalla definizione di oggetti che rappresentano le parti del discorso, come verbi, soggetti, oggetti e modificatori.
Parsificando le informazioni contenute nei SentencePlans generati al punto precedente in formato JSON, è possibile creare un oggetto in cui le informazioni estratte possano essere utilizzate da SimpleNLG per la costruzione della traduzione.
Nel caso siano presenti più complementi oggetto o più soggetti, SimpleNLG è in grado di gestire la coordinazione tra più elementi dello stesso tipo.
Esempio traduzione:

```
Dante and a woman give a gift to Virgilio.
```

Translation RealTime
Queste differenti fasi di generazione automatica di linguaggio naturale sono state inserite in una pipeline in modo da tradurre “al volo” frasi che possono essere descritte con la stessa grammatica delle frasi di esempio.
Utilizzo:

```
> ./Translate.sh “un uomo ama Francesca”
> A man loves Francesca
```

Punto Bonus: estrarre una PCFG dal Penn-TUT e provarla in NLTK con le frasi di esempio.

NLTK fornisce il treebank Penn su cui è possibile estrarre le regole di produzione e creare una Context Free Grammar.
Utilizzando la formula `Count(A → B)/Count(A)` per descrivere la probabilità della regola `A → B,` è stata creata una nuova grammatica probabilistica contenente le informazioni statistiche.
A causa della presenza di caratteri non alfanumerici utilizzati come simboli non terminali, `nltk.parse_pcfg()` produce eccezioni durante la generazione della grammatica; per questo motivo sono state necessarie innumerevoli `replace` all'interno delle regole di produzione.
Le frasi di esempio non sono parsificabili perchè mancano le definizioni dei termini all'interno del treebank, quindi ho utilizzato un'altra frase come esempio.
L'albero sintattico prodotto dalla parsificazione della frase di test non è corretto, ma è stato generato utilizzando un'oracolo probabilistico.

```
I can finally drink a beer now
```

```
(PP-PRD
  (NP-ADV
    (NP (NP (PRP I)) (NN can))
    (PP (RB finally) (NP (NN drink))))
  (IN a)
  (NP (NN beer) (RB now))) (p=1.86499094394e-39)
```