TECNOLOGIE DEL LINGUAGGIO NATURALE

Traduzione Interlingua EN $\rightarrow$ IT

Esercizio 3

Autore:
Delsanto Matteo
github.com/TeoMatt

24 Luglio 2018

Indice

Sommario

Lo scopo di questo esercizio è l'analisi dell'inglese mediante la libreria *NLTK* e la generazione della traduzione in italiano mediante la libreria *SimpleNLG-it*. Per fare ciò si proceduto per prima cosa alla stesura di una grammatica, interpretabile da *NLTK*, che potesse parsificare le frasi in input. Il risultato del parsing utilizzando questa grammatica, dotata di semantica, è una espressione logica che rappresenta, appunto, la semantica della frase in input. A partire da questa espressione si è adottato un approccio basato su espressioni regolari per il riconoscimento degli elementi della frase e una lessicalizzazione EN $\rightarrow$ IT 1-1 che associa ad ogni senso presente nell'espressione logica una parola italiana. Gli elementi e le relazioni tra di essi vengono usati per la creazione di un albero che rappresenta il *sentence plan*. Ogni elemento del sentence plan viene elaborato e unito agli altri per la realizzazione della frase italiana tramite la libreria *SimpleNLG-it*.

Capitolo 1

La grammatica

In questo capitolo si analizza la grammatica utilizzata con *NLTK* per la parsificazione delle frasi.

1.1 Grammar

La grammatica utilizzata permette la parsificazione di frasi del tipo:

```
[Aggettivo +] Soggetto [+ ComplSpec +] [ + Neg +] Verbo_Essere [+ Avverbio +] Aggettivo

[Aggettivo +] Soggetto [+ ComplSpec +] [ + Neg +] Verbo_Make [+ Aggettivo +] ComplOggetto Aggettivo

[Aggettivo +] Soggetto [+ ComplSpec +] [ + Neg +] Verbo_Use [+ Aggettivo +] ComplOggetto [+ Aggettivo +] ComplScopo
```

Per quanto riguarda la semantica associata, vediamo nel dettaglio le scelte di rappresentazione: i sostantivi e gli aggettivi usati non congiuntamente ad un nome sono rappresentati come formula del tipo *P(x)* (ad esempio per mind troveremo *mind(x)* e per great *great(x)*). Per differenziare i sostantivi singolari e plurali si è scelto di allegare alla formula la stringa "_pl" per i plurali (per minds troveremo *mind_pl(x)*). Gli aggettivi che invece compaiono strettamente collegati al sostantivo (great mind) compariranno come formule del tipo *P(Q(x))* (ad esempio *great(mind(x))*). Gli articoli dichiarano l'esistenza di una variabile e due formule *P(x)* e *Q(x)*, una che verrà definita dal sostantivo di riferimento e una dal verbo. Se gli articoli indeterminativi si limitano a fare questo, i determinativi invece aggiungono informazione, andando ad affermare che si sta parlando proprio di una variabile e di una formula specifici e riconoscibili, come d'altronde è loro semantica in una qualsiasi frase.

```
Det[NUM=pl,SEM=<\P Q.exists x.((P(x) & Q(x)) & all y.(P(y) -> (x = y)))>] -> 'the'
Det[NUM=sg,SEM=<\P Q.exists x.(P(x) & Q(x))>] -> 'a'

N[NUM=sg,PER=tre,SEM=<\x.mind(x)>] -> 'mind'
N[NUM=pl,PER=tre,SEM=<\x.mind_pl(x)>] -> 'minds'
```

```
Adjn[NUM=?n,SEM=<\P x.great(P(x))>] -> 'great'
Adj[NUM=sg,SEM=<\x.great(x)>] -> 'great'
```

Per quanto riguarda i verbi, essi vengono rappresentati come formule con arietà diverse a seconda del verbo e dei complementi associati: il verbo *be* è rappresentato come l'applicazione di una formula e non aggiunge nulla alla semantica (*\P x.P(x)*). il verbo *make*, denominato come transitivo, è rappresentato da una formula che prende in input la variabile che compie l'azione (soggetto) e chi la subisce, con applicata la nuova formula che rappresenta la trasformazione propria di questo senso del verbo *make* (*make(x, P(x))*): nella frase 'the mind makes a jedi great', avendo *mind(x)* e *jedi(y)*, otterremo *make(x,great(y))*. Il verbo *use*, denominato com doppiamente transitivo, necessita di un parametro in più perchè oltre al complemento oggetto è necessario specificare anche il complemento di scopo. in forma generica avremo quindi *use(x,y,z)*. Un esempio: "the jedi uses the force for knowledge" avrà una formalizzazione del verbo *use* che, dati *jedi(x)*, *force(y)*, *knowledge(z)*, sarà *use(x,y,z)*. Ovviamente questa formalizzazione può essere generalizzata ed estesa con altri verbi.

```
Be[NUM=sg,PER=tre,SEM=<\P x.P(x)>,tns=pres] -> 'is'

TV[NUM=sg,PER=tre,SEM=<\Y X \y.Y(\z.make(y,X(z)))>,TNS=pres] -> 'makes'

DTV[NUM=sg,PER=tre,SEM=<\Y X x.X(\z.Y(\y.use(x,y,z)))>,TNS=pres] -> 'uses'
```

Per quanto riguarda il terminale for, la sua semantica viene inglobata nel verbo *use*, mentre per il terminale of, esso aggiunge il complemento di specificazione e congiunge quindi la NP alla VP tramite una semantica del tipo *\x.(of(x,y) & Q(y))*.

```
P[+for] -> 'for'

O[SEM=<\Y X Q (X(\y.Y(\x.(of(x,y) & Q(y)))))>] -> 'of'
```

Le negazioni sono inserite come dei simboli - legati o direttamente ad una formula, come nel caso degli aggettivi attribuiti tramite verbo *be* (is great → *great(x)*, is not great → *-great(x)*), oppure prima delle dichiarazioni delle variabili usate nei verbi, per negare quindi il verbo stesso (soggetto x non fa una persona grande → *-exists z1.(one(z1) & make(x,great(z1)))*).

Ogni regola di riscrittura è associata però, come visibile dagli esempi, non solo ad una semantica, ma anche ad altri due parametri (numero e persona) che permettono un controllo più puntuale sulla parsificazione, identificando eventuali errori di accordo fra plurali e singolari e di coniugazione dei verbi in base alla persona. Durante la costruzione dell'albero vengono infatti scelti i terminali che, oltre a corrispondere alla stringa inserita, si accordano tra di loro in numero e persona. Facciamo un esempio: nella frase 'The mind is great' il primo controllo avviene tra the

e mind, i quali si accordano in numero con la scelta del terminale mind, singolare, e del terminale the con la feature *NUM* a singolare. Il secondo controllo viene invece effettuato per la scelta del verbo nella Verbal Phrase: le feature *NUM* e *PER* di VP dipendono infatti esclusivamente dall'elemento Verbal, che in questo caso conterrà il terminale *is* il quale risulta coniugato alla terza persona singolare (mind è proprio alla terza persona singolare). Un eventuale errore nella scrittura della frase, come per esempio 'the mind are great', comporterebbe l'impossibilità di trovare un albero di parsificazione corretto, in quanto *are* rappresenta una coniugazione del verbo che che potrebbe accordarsi sulla feature *NUM* al singolare ma che risulterebbe comunque alla seconda persona, impossibile quindi da accordare con la parola mind (terza persona singolare).

In più, rispetto alle frase proposte, si è pensato di inserire la gestione dell'ausiliare *do* per la negazione. Ci si limita alla lettura dell'ausialiare, coniugato correttamente, e al controllo della giusta coniugazione del verbo principale, il quale, in presenza di ausiliare, deve presentarsi in forma base per qualsiasi persona (per comodità viene forzata la coniugazione ad una persona plurale, uguale alla forma base).

```
Aux[NUM=sg,PER=uno,SEM=<\P x.P(x)>,tns=pres] -> 'do'

Aux[NUM=sg,PER=tre,SEM=<\P x.P(x)>,tns=pres] -> 'does'
```

Ecco alcuni esempi di regole di riscrittura:

```
S[SEM = <?subj(?vp)>] -> NP[NUM=?n,PER=?p,SEM=?subj] VP[NUM=?n,PER=?p,SEM=?vp]

NP[NUM=?n,PER=?p,SEM=<?nominal>] -> Nominal[NUM=?n,PER=?p,SEM=?nominal]
NP[NUM=?n,PER=?p,SEM=<app(?nominal,?subj)>] -> Nominal[NUM=?n,PER=?p,SEM=?subj] C[+OF, SEM=?nominal]

VP[NUM=?n,PER=?p,SEM=<?a(?neg(?v))>] -> Aux[NUM=?n,PER=?p,SEM=?a] Neg[SEM=?neg] Verbal[NUM=pl,PER=?p,SEM=?v]
VP[NUM=?n,PER=?p,SEM=<?neg(?v)>] -> Neg[SEM=?neg] Verbal[NUM=?n,PER=?p,SEM=?v]
VP[NUM=?n,PER=?p,SEM=<?v>] -> Verbal[NUM=?n,PER=?p,SEM=?v]
```

La grammatica completa è comunque visibile su GitHub al link github.com/TeoMatt

Capitolo 2

Il Traduttore

In questo paragrafo analizzeremo il vero e proprio traduttore.

2.1 Python e NLTK

La parte di codice che utilizza la grammatica è scritta in *python* e semplicemente utilizza il parser fornito dalla libreria *NLTK*, in grado di verificare la correttezza della frase data in input e di estrapolarne la semantica unendo quella dei singoli terminali e delle regole di riscrittura man mano che viene costruito l'albero di parsing. Il risultato della parsificazione, in logica del prim'ordine, è una stringa, la quale viene passata in input al vero e proprio traduttore tramite uno script .sh .

Listing 2.1: Codice Python

```
1 import nltk
2 from nltk import load_parser
3 cp = load_parser('ex3_per.fcfg')
4 query = 'query'
5 trees = list(cp.parse(query.split()))
6 answer = trees[0].label()['SEM']
7 print(answer)
```

2.2 Kotlin e SimpleNLG-IT

Il traduttore si basa sull'utilizzo di espressioni regolari che analizzano la stringa individuando i costituenti della frase. Questi vengono poi tradotti in italiano con una lessicalizzazione 1-1 e posizionati in una struttura *json* che ne rappresenta il sentence plan. Dato il sentence plan, i vari elementi sono composti tra di loro secondo le regole della morfologia e sintassi italiana grazie alla libreria *SimpleNLG-IT*.

Scendiamo ora più nel dettaglio ed analizziamo i punti salienti del traduttore:
innanzitutto viene inizializzato l'oggetto json che ospiterà il sentence plan, *rootObject*, e i due sotto oggetti principali che rappresentano le relazioni di *nsubj* e *obj*. Nella prima parte del codice, viene analizzato il soggetto della frase che, nella stringa in input, comparirà come prima variabile dichiarata. L'espressione regolare individua quindi la prima *P(x)* (sostantivo) o *P(Q(x))* (sostantivo con aggettivo). Attraverso una mappa che lega ogni parola inglese alla sua traduzione si individua la parola italiana e la si utilizzata per inserirla nel *nsubj* con chiave val e l'eventuale aggettivo viene settato nel campo *amod*. Una successiva espressione regolare si occupa di impostare a *true* o *false* il campo *det*, andando a verificare la presenza o meno della semantica *& all y.(P(y) -> (x = y))* con *P* soggetto, tipica dell'articolo determinativo *the*.

Listing 2.2: Soggetto con Aggettivo

```
1 adjective = totrim.split("""exists ${varmap["subject"]}\.\(""".toRegex())[1].split("""\([a-z]+_*[a-z
      ]+\(${varmap["subject"]}\).*""".toRegex())[0]
2 subject = totrim.split("""exists ${varmap["subject"]}\.\([a-z]+\(""".toRegex())[1].split("""\(${varmap
      ["subject"]}\)""".toRegex())[0]
3 nsubj.put("val", map[subject])
4 nsubj.put("amod", map[adjective])
5 if (logicExpression.matches("""*& all .*\.\($adjective\($subject\(.*\)\) -> \(${varmap["subject"]}
      =.*""".toRegex())) {
6       nsubj.put("det", "true")
7 }else{
8       nsubj.put("det", "false")
9 }
```

A questo punto, il successivo if-case si occupa di differenziare le azioni per verbo *be* o verbi *make/use* (o presenza di complemento di specificazione) in base a alla presenza o meno di successive dichiarazioni di variabili. Nel caso del verbo *be* (nessuna dichiarazione di variabile) lo si setta come *val* del *rootObject* e si individua la nuova formula dichiarata *P(x)* (aggettivo) o *Q(P(x))* (avverbio + aggettivo). L'aggettivo sarà inserito nel campo *scomp* e l'eventuale avverbio nel campo *adv_scomp* dell'oggetto *obj*. In questo caso non ci saranno articoli. Nel caso ci sia una successiva dichiarazione di variabile con due formule successive, allora potremmo trovarci nel caso del verbo *make* (che prende in input due parametri) o di un complemento di specificazione, che vedremo in seguito. Nel caso del verbo *make*, o più in generale di un verbo transitivo, avremo come primo parametro il soggetto e come secondo il complemento oggetto. Si analizza proprio il complemento oggetto che apparirà nella forma *P(x)* con *P* aggettivo e *x* l'ultima variabile dichiarata con relativa formula. Si procederà quindi l'inserimento del verbo nel campo *val* del *rootObject* e con l'inserimento del sostantivo e dell'aggettivo rispettivemente nei campi *val* e *amod_p* dell'*obj*. Un discorso simile vale per il verbo *use*, o più in generale per i verbi "doppiamente transitivi" che prendono in input il soggetto, il complemento oggetto e il complemento di scopo. In questo caso, dopo la seconda dichiarazione di variabile, ce ne sarà una terza, con relativa formula, seguita dal verbo con i suoi parametri. Avverrà quindi come per il verbo *make* l'inserimento del verbo nel campo *val* del *rootObject* e l'inserimento del

sostantivo nel campo *val* dell'*obj*. La novità sta nell'utilizzo del campo *prep* dell'*obj* che conterrà la dipendenza del complemento di scopo, inserendo i campi *p* per la preposizione, *noun* per il sostantivo, *plural*, *det* e eventuale *amod*

Listing 2.3: Esempio gestione verbo *make*

```
clause.setVerb(map[tv])
rootObject.put("val", map[tv])
obj.put("val", map[propvar1])
if(aggvar1 != ""){
       obj.put("amod", map[aggvar1])
}
if (logicExpression.matches("""*& all [a-z][0-9]*\.\($propvar1\(.*\) -> \($variabile1 =.*""".toRegex
     ()) || logicExpression.matches("""*& all [a-z][0-9]*\.\($aggvar1\($propvar1\(.*\) -> \(
     $variabile1 =.*""".toRegex())){
       obj.put("det", "true")
}else{
       obj.put("det", "false")
}
obj.put("amod_p", map[propervar1])
if (daNegare) rootObject.put("neg", "true") else rootObject.put("neg", "false")
if (propvar1.matches("""*_pl""".toRegex())) obj.put("plural", "true") else obj.put("plural", "false")
```

Per quanto riguarda il complemento di specificazione, essendo una formula con due parametri, deve essere controllato esplicitamente e differenziato da un normale verbo transitivo. La sua gestione prevede l'uso del campo *prep* del *subj*, con i campi *p* per la preposizione, *noun* per il sostantivo, *plural*, *det* e eventuale *amod*. Dopo il complemento di specificazione si possono verificare tutti i casi descritti in precedenza (verbo be, transitivo o "doppiamente transitivo") e quindi il codice svolge controlli del tutto analoghi ai precedentemente descritti. Per ognuna di queste casistiche viene anche controllato il tipo degli articoli, con lo stesso controllo proposto per il soggetto. Si nota, infine, che sono possibili eventuali negazioni che, nel caso del verbo be, vengono gestite con un flag *negato* settato coerentemente alla presenza del ' - ' direttamente davanti alla formula da negare, mentre nel caso degli altri verbi vengono gestite con un secondo flag *daNegare*, ma settato coerentemente alla presenza del ' - ' prima della dichiarazione di esistenza delle variabili che occorreranno nei verbi di interesse.

Listing 2.4: Negazione con verbo be

```
var negato = false
var aggettivo: String
var avverbio = ""
if (totrim.matches("""-*[a-z]+\([a-z]+\(${varmap["subject"]}\)\).*""".toRegex())){
       if (totrim.matches("""-.*""".toRegex())) {
              totrim = totrim.split("""-""".toRegex())[1]
              avverbio = totrim.split("""\(""".toRegex())[0]
              aggettivo = totrim.split("""\(""".toRegex())[1]
              negato = true
       }else{
              avverbio = totrim.split("""\(""".toRegex())[0]
              aggettivo = totrim.split("""\(""".toRegex())[1]
```

```
13          }
14  }else{
15          if (totrim.matches("""-.*""".toRegex())) {
16                  totrim = totrim.split("""-""".toRegex())[1]
17                  aggettivo = totrim.split("""\(""".toRegex())[0]
18                  negato = true
19          }else{
20                  aggettivo = totrim.split("""\(""".toRegex())[0]
21          }
22  }
```

L'ultima operazione da effettuare è impostare i due oggetti nsubj e obj nei campo omonimi del rootObject e richiamare il metodo realize che, con l'ausilio di SimpleNLG-IT genera la frase italiana.

2.3 Realize

Il metodo *realize* analizza semplicemente l'esistenza e i valori dei campi dell'oggetto json in input e genera la frase italiana di conseguenza. Viene creata una *clause* di cui viene effettuata la *setVerb* con il verbo inserito nel campo *val* del *rootObject*. La *clause* avrà poi come *subject* una *AdjectivePhrase* con l'avverbio come *preModifier*, nel caso del verbo be, o una *NounPhrase* con adatto *specifier* (articolo) in base al campo *det* del *nsubj*, negli altri casi. L'*object* della *clause* sarà invece sempre una *NounPhrase*. Vengono infine controllati gli eventuali campi *prep* di *nsubj* e *obj* e quindi all'occorrenza create delle *PrepositionPhrases* legate alle frasi soggetto o complemento oggetto come *postModifiers*. L'ultimo step è la realizzazione della frase tramite il metodo *realiseSentence(clause)*.

Listing 2.5: esempio json

```
1   {
2           "val": "essere",
3           "neg": "false",
4           "obj": {
5                   "scomp": "grande",
6                   "adv_scomp": "veramente"
7           },
8           "nsubj": {
9                   "val": "mente",
10                  "det": "true",
11                  "plural": "false",
12                  "prep": {
13                          "p": "di",
14                          "det": "false",
15                          "plural": "false",
16                          "noun": "bambino"
17                  }
18          }
19  }
```