Università degli Studi di Torino
Corso di Laurea Magistrale in Informatica
Tecnologie del Linguaggio Naturale
Docente: Alessandro Mazzei

Traduttore ENG - ITA

Componenti del Gruppo
- Alessandro CLOCCHIATTI (matricola 909105);
- Michele PELLEGRINO (matricola 909121).

---

1. Introduzione

La seguente relazione illustra la realizzazione di un traduttore interlingua da inglese ad italiano.
L'esercizio consiste nel parsificare ed interpretare le seguenti frasi inglesi:
- *You are imagining things*.
- *There is a price on my head*.
- *Your big opportunity is flying out of here*.

tradurle e infine costruire, tramite l'API SimpleNLG-it [2], una proposizione sintatticamente corretta.

In particolare, la consegna prevede i seguenti step:
1. Scrivere una grammatica CF con semantica (ispirarsi alla grammatica *simple-sem.fcfg*, grammars/book_grammars [3][4]) per le frasi in ingresso e usare NLTK [2] per parsificare;
2. Costruire un sentence planner che per ogni formula logica prodotta dalla grammatica crei un sentence plan valido per SimpleNLG-it;
3. Usare una lessicalizzazione più semplice possibile per trasformare le costanti e i predicati in parole italiane;
4. Usando la libreria SimpleNLG-IT implementare un realizer che trasformi i sentence plans in frasi italiane [2].

La relazione inizialmente illustra la costruzione della grammatica e la sua parsificazione, successivamente viene spiegato il funzionamento del sentence planner e della lessicalizzazione delle costanti e dei predicati e infine si descrive lo script che permette di realizzare le frasi tradotte a partire dai sentence plan e vengono proposti alcuni miglioramenti futuri.

2. Costruzione della grammatica

Partendo dalle frasi in input abbiamo costruito una grammatica context-free arricchita con una semantica basata sulla logica del primo ordine. La grammatica utilizzata è composta da una serie di simboli non terminali, che rappresentano i costituenti della frase (NP, VP, PP), e simboli terminali dotati di una rappresentazione semantica e alcune features utili per il processo di traduzione. Nella seguente tabella vengono illustrate le features usate.

Espressione	Significato
LOC	Espressioni locative
POSS	Espressioni possessive
PERS	Pronomi personali
NUM	Numero (singolare o plurale)
GEN	Genere (maschile o femminile)
TNS	Tempo verbale

L'utilizzo di tali features si è rivelato essenziale per gestire le parole e nell'utilizzare l'API SimpleNLG [1]. Data la particolarità di "*There is*" è stata introdotta una categoria sintattica EX che rappresenta la componente *There* la cui semantica si lega a quella del verbo che segue (*is*). È stato introdotto anche un tag CP per rappresentare una copula e AUX per i verbi ausiliari. La loro semantica sarà del tipo $\lambda x.x$ in quanto privi di significato proprio.

2.1 Frasi con verbi transitivi

La costruzione semantica più immediata è proprio quella dei verbi transitivi, costituita da soggetto, verbo e oggetto (struttura predicato-argomento molto semplice da rappresentare in F.O.L.). La formalizzazione proposta è costituita da un predicato $verb(soggetto, oggetto)$ e da un predicato $obj(x)$ che specifica l'oggetto.

$$exists\ x.(obj(x), verb(subj, x))$$

La frase in oggetto è *You are imagining things* e la semantica in first order logic che otterremo dal nostro sistema sarà:

$$exists\ z2.(thing(z2)\ \&\ image(you, z2))$$

La regola di produzione *S*, radice della frase, è composta da una frase nominale (NP) ed una frase verbale (VP):

$$S[SEM\ = <?\ subj(?\ vp)>] \rightarrow\ NP[NUM =?\ n, SEM =?\ subj]\ VP[NUM =?\ n, SEM =?\ vp]$$

Per i verbi transitivi è stata scelta una rappresentazione del tipo $verb(x, y)$ dove x indica il soggetto e y l'oggetto.

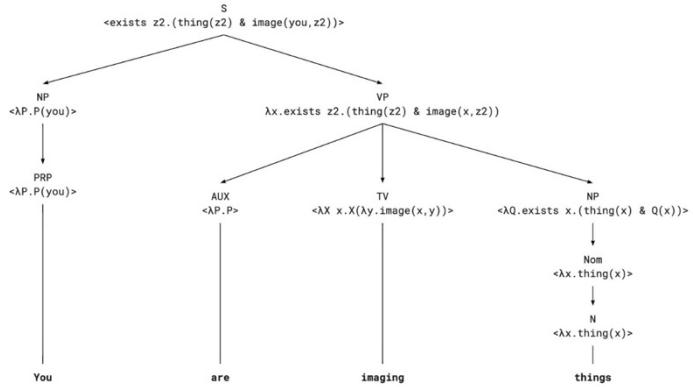

Figura 1. Albero di derivazione della frase "You are imagining things"

2.2 Frasi con verbi intransitivi

La formalizzazione proposta per le frasi con i verbi transitivi è costituita da un predicato che rappresenta la forma verbale $VP(e)$ e da un'altra costruzione che comprende soggetto e complemento della frase:

$$exists\ x.(exists\ e.(VP(e)\ \&\ exists\ z.(subj(x), complement(x,z))))$$

Una delle due frasi in input è *There is a price on my head* e la semantica in first order logic che otterremo dal nostro sistema sarà:

$$exists\ x.(exists\ e.(presence(e)\ \&\ agent(e,x))\ \&\ exists\ z5.(my(z5)\ \&\ head(z5)\ \&\ price(x)\ \&\ on(x,z5)))$$

La terza frase in input invece è *Your big opportunity is flying out of here* che avrà come formalizzazione:

$$exists\ x.(subj(x), exists\ e.(VP(e), exists\ y.(Pred(e,y))))$$

e l'output del sistema invece sarà:

$$exists\ x.(your(x)\ \&\ big(x)\ \&\ opportunity(x)\ \&\ exists\ e.(fly(e)\ \&\ agent(e,x)\ \&\ out(e)\ \&\ exists\ y.(from(e,y)\ \&\ here(y))))$$

Poiché nelle frasi con costruzione verbale intransitiva analizzate, erano presenti sia avverbi che complementi, per poterli descrivere abbiamo deciso di utilizzare una rappresentazione Neo-Davidsoniana per i verbi intransitivi, ottenendo dunque una costruzione semantica del tipo: $verb(e), agent(e,x), adverb(e)$. Sia gli avverbi che gli aggettivi vengono associati ad un predicato del tipo: $\lambda x.adj(x)$. Infine, attraverso il type raising abbiamo gestito i termini che rappresentano i soggetti nella forma $\lambda P.P(term)$. Di seguito la costruzione semantica del verbo *flying* presente nella terza frase. Lo abbiamo considerato nella sua forma già coniugata perché salviamo nella struttura delle features il tempo verbale della parola, che sarà poi utilizzato da SimpleNLG per la costruzione della frase.

$$IV\big[NUM = sg, SEM = <\backslash P\ T\ x.exists\ e.\big(fly(e)\&\ agent(e,x)\&\ T(e)\&\ P(e)\big)>, TNS = ger\big] \rightarrow\ 'flying'$$

Inoltre, proprio per la presenza dell'espressione *There is* all'inizio di una delle frasi abbiamo dovuto differenziare la regola di produzione S di questo tipo di frase, in questo modo:

$$S[SEM\ = <?\,vp(?\,np)>] \rightarrow\ VP[NUM = ?\,n, SEM = ?\,vp]\ NP[NUM = ?\,n, SEM = ?\,np]$$

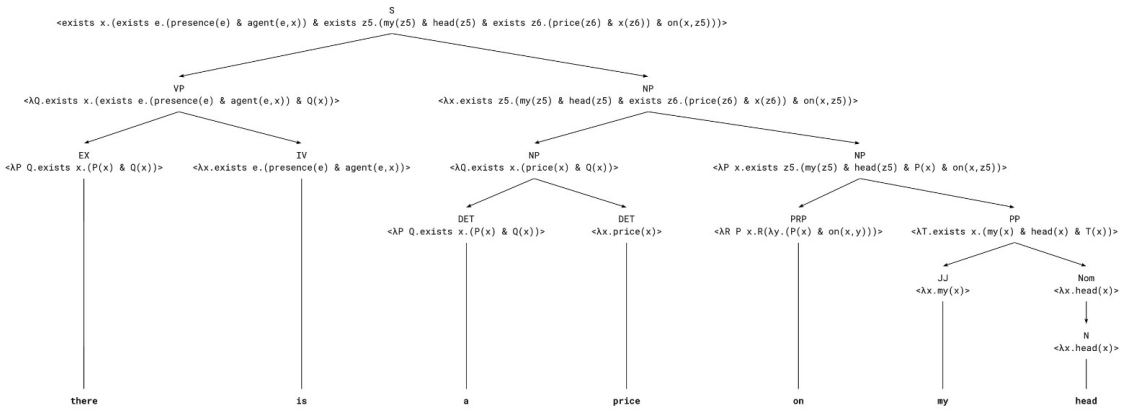

Figura 2. Albero di derivazione della frase "There is a price on my head"

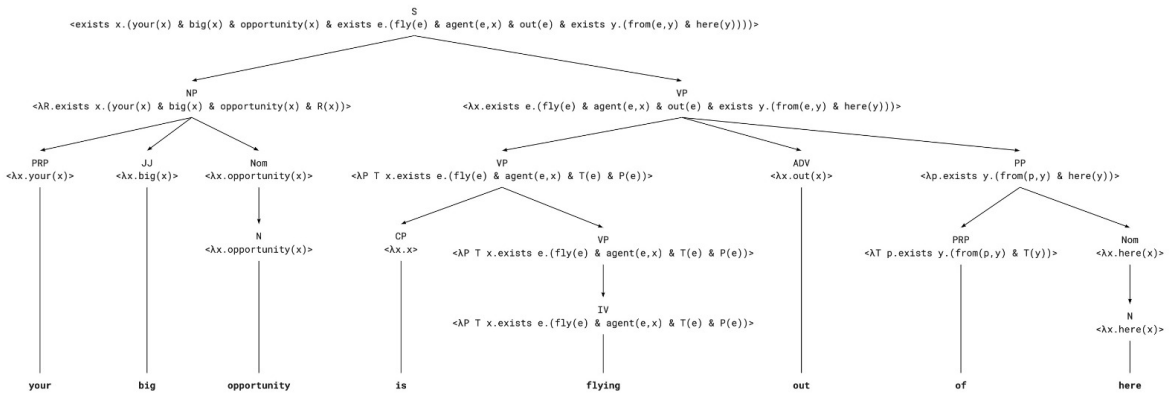

Figura 3. Albero di derivazione della frase "Your big opportunity is flying out of here"

3. Parsificazione

Dopo aver definito la grammatica è possibile effettuare la parsificazione delle frasi in input attraverso la funzione `parseSentence()`, la quale sfrutta il metodo NLTK `parse()` per parsificare la frase in base alla grammatica creata.

```
parser = load_parser(grammar, trace=0)
trees = list(parser.parse(sentence.split()))
```

Di seguito gli alberi sintattici riferiti alle tre frasi analizzate:

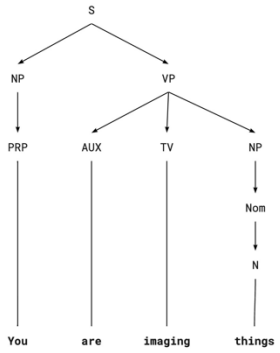

Figura 4. Albero sintattico della frase "You are imaging things"

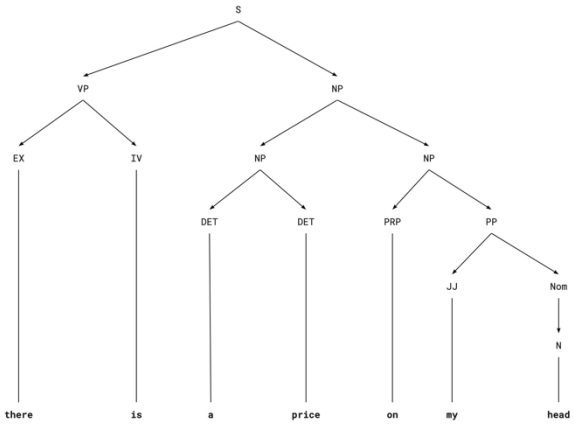

Figura 5. Albero sintattico della frase "There is a price on my head"

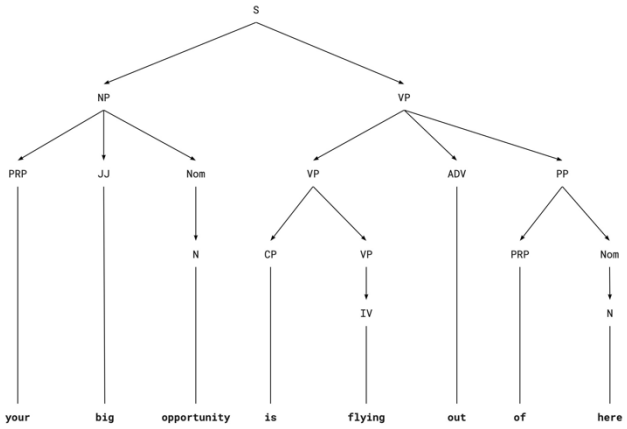

Figura 6. Albero sintattico della frase "Your big opportunity is flying out of here"

Dopo aver determinato l'albero, la formula semantica viene determinata tramite la funzione NLTK `tree.label()['SEM'].simplify()`, la quale permette di determinare le espressioni logiche. L'attributo `simplify()` esegue la β-reduction sulle varie espressioni.

4. Sentence planner

In base alla rappresentazione semantica delle frasi proposte, è possibile identificare tre classi:

- $exists\ x.(obj(x), verb(subj, x))$, ovvero una frase dichiarativa con soggetto, verbo transitivo e oggetto;

- $exists\ x.(exists\ e.(VP(e), exists\ z.(subj(x), complement(x, z))))$, ovvero una frase con soggetto, verbo e complemento, il quale mette in relazione la variabile del soggetto (x) con un NP;

- $exists\ x.(subj(x), exists\ e.(VP(e), exists\ y.(Pred(e, y))))$, ovvero una frase con soggetto, verbo predicato che mette in relazione la variabile del verbo (e) con un NP.

In base al template della formula è possibile determinare gli elementi della frase.
Inizialmente viene determinato il template dell'espressione logica (`formula`) con la funzione `getFormulaTemplate()`, la quale effettua un'operazione di matching tra le formule e l'espressione regolare. Inoltre vengono determinati tutti i predicati semantici della formula con la funzione `getSemanticTerms()`, che effettua una ricerca ricorsiva tra gli elementi dell'espressione. In particolare, nel caso in cui il termine sia un esistenziale (`ExistsExpression`) viene eseguita la ricerca sui termini che la compongono (`term.term`). Nel caso in cui il termine sia un'espressione And (`AndExpression`), ovvero nel caso in cui abbia gli attributi `first` e `second`, viene fatta la ricerca sui rispettivi predicati. Infine, nel caso in cui il termine abbia un attributo `pred`, ovvero nel caso in cui si tratti un nodo foglia, il termine viene aggiunto alla lista dei predicati della formula.

Dopo aver determinato il template e tutti i predicati della formula, il procedimento continua affrontando le varie casistiche degli argomenti e quindi andando ad assegnare correttamente gli argomenti (soggetto, oggetto, oggetto, complemento).

4.1 Template 0

$$exists\ x.(obj(x), verb(subj, x)),$$

In questo caso il verbo corrisponde all'ultimo predicato della formula e, sapendo che si tratta di un verbo transitivo, i suoi argomenti (`subj` e `x`) vengono determinati con la funzione `transitiveVerbArguments()`. Quest'ultima ritorna la lista dei nomi degli argomenti del predicato verbo. Mentre l'argomento corrispondente al soggetto (`subj`) risulta essere il nome del predicato, l'argomento dell'oggetto è una variabile (`vobj`). Per questo motivo risulta necessario cercare tutti i predicati che contengono `vobj` (attraverso la funzione `findOccurencies()`).

```
def findOccurencies(tree, terms, var):
    pred = set()
    res = []

    for t in terms: # for each formula terms
        args = set(map(lambda x: x.variable.name, t.args))
        if var in args:
            pred.add(t)

    for p in pred:
        node = add_pred_pos(tree, p)
        if node is not None and not res.__contains__(node):
            res.append(node)

    return res
```

La funzione `findOccurencies()` seleziona, tra tutti i predicati della formula, quelli i cui argomenti corrispondono alla variabile (`var`) da cercare. Successivamente, ad ogni predicato che soddisfa la precedente condizione, vengono aggiunte le corrispondenti informazioni sintattiche (part-of-speech tags). Questa operazione viene fatta con la funzione `add_pred_pos()`, la quale inizialmente determina il nome del predicato, calcola tutti i possibili sottoalberi dell'albero iniziale (`tree`) e, tra tutti i sotto-alberi, seleziona solo quelli che risultano essere delle foglie in base ai simboli terminali che sono stati utilizzati nella grammatica:

```
['TV', 'IV', 'DTV', 'N', 'JJ', 'PropN', 'Det', 'EX', 'PRP', 'AUX', 'CP', 'ADV']
```

```
def add_pred_pos(tree, term):
    leaves = ['TV','IV','DTV','N','JJ','PropN','Det','EX','PRP','AUX','CP','ADV']
    pred_name = term.pred.variable.name if hasattr(term,'pred') else term

    terminals = [i for i in tree.subtrees()]  # get all subtrees

    terminals = list(
        filter(lambda x: re.search(
             "\\'(.*)\\'", str(x.label()).split('\n')[0], re.IGNORECASE).group(1)
             in leaves,terminals))

    for t in terminals:
        if pred_name == getPredicateLemma(t):
            tag = re.search("\\'(.*)\\'",
                    str(t.label()).split('\n')[0], re.IGNORECASE).group(1)

            node = {'pred': pred_name, 'tag': tag}

            # [*type*, 'NUM', 'SEM', 'TNS', 'GEN']
            if 'NUM' in t.label().keys():
                node['num'] = t.label()['NUM']
            if 'TNS' in t.label().keys():
                node['tns'] = t.label()['TNS']
            if tag in ['N', 'PropN'] and 'GEN' in t.label().keys():
                node['gen'] = t.label()['GEN']
            if 'LOC' in t.label().keys():
                node['loc'] = True

            return node
    return None
```

Per ogni foglia, nel caso in cui il nome del predicato iniziale corrisponda al lemma del predicato del nodo foglia, viene creato un dizionario contenente alcune informazioni relative alla sintassi e alla semantica (*nome del predicato, POS tag, numero, genere, tempo verbale, …*), ottenendo il seguente risultato

$$\{'pred': 'image', 'tag': 'TV', 'num': 'sg', 'tns': 'ger'\}$$
$$\{'pred': 'thing', 'tag': 'N', 'num': 'pl', 'gen': 'f'\}$$
$$\{'pred': 'you', 'tag': 'PRP', 'num': 'sg'\}$$

L'unione tra le informazioni sintattiche e semantiche risulta essere utile in quanto, sapendo che l'oggetto del verbo ha come part-of-speech tag N (nome), è possibile selezionare dalla lista di predicati contenenti la variabile oggetto (`vobj`) solo quello il cui tag corrisponde al nome.
Infine, come fatto con il predicato oggetto, vengono aggiunte le informazioni sintattiche ai predicati verbo e soggetto, attraverso la funzione `add_pred_pos()`.

4.2 Template 1

$$exists\ x.(exists\ e.(VP(e), exists\ z.(subj(x), complement(x, z))))$$

In questo caso il verbo corrisponde al primo predicato nella formula e, sapendo che si tratta di un verbo intransitivo, il soggetto viene determinato con la funzione `intransitiveVerbSubj()`. Questa funzione seleziona tra i termini semantici il predicato il cui nome è "*agent*". Il predicato in cui è presente il soggetto risulta essere il primo e, la variabile del soggetto (`vsubj`), è il secondo argomento del predicato. Per determinare il nome del predicato soggetto è necessario cercare tra tutti i predicati della formula quelli che contengono la variabile `vsubj` (funzione `findOccurencies()`). Il predicato corrispondente al soggetto risulta essere quello il cui tag sintattico è N (nome). Gli altri predicati in cui compare la variabile soggetto sono, invece, modificatori del soggetto (`mod_subj`).
Dopo aver determinato il soggetto, vengono determinate le informazioni sintattiche riferite al predicato verbo.

Per determinare il predicato corrispondente al complemento, si calcolano inizialmente tutte le variabili presenti nella formula (funzione `getAllVariables()`). Dalla lista totale di variabili vengono rimosse quelle corrispondenti al verbo e al soggetto (`getAllVariables(semTerms) - variablesVisited`). Il complemento sarà composto da una lista di dizionari riferiti ai predicati in cui la variabile complemento compare (attraverso `findOccurencies()`).

I predicati in cui compare il verbo ma che non risultano essere dei complementi saranno invece i modificatori dei verbi (`mod_verb`).

4.3 Template 2

$$exists\ x.(subj(x), exists\ e.(VP(e), exists\ y.(Pred(e,y))))$$

In questo caso il predicato del verbo (`verb_pred`) corrisponde al quarto predicato della formula. Sapendo che il verbo è intransitivo, il soggetto viene determinato con la funzione `intransitiveVerbSubj()`, la quale permette di stabilire la variabile associata al soggetto andando a selezionare tra i termini semantici quelli il cui nome è "*agent*". Il predicato in cui è presente il soggetto è il primo e la variabile soggetto (`vsubj`) risulta essere il secondo argomento del predicato.
Per determinare poi il predicato associato al soggetto è necessario cercare tra tutti i predicati della formula quelli che contengono la variabile `vsubj` tramite la funzione `findOccurencies()`. Il predicato il cui tag sintattico è N (nome) è quindi quello riferito al soggetto, mentre gli altri sono modificatori.

Per determinare i modificatori del verbo si estrae la variabile `event` dal predicato `verb_pred`. Successivamente si determinano tutti i predicati in cui compare `event` attraverso la funzione `findOccurencies()`, determinando così i modificatori del verbo (`mod_verb`). Infine al predicato del verbo vengono aggiunte le informazioni sintattiche attraverso la funzione `add_pred_pos()`.

Per determinare il complemento, come nel caso precedente (*template 1*), inizialmente vengono determinate tutte le variabili presenti nella formula (`getAllVariables()`) e, tra queste, vengono rimosse quelle corrispondenti al verbo e al soggetto (`getAllVariables(semTerms) - variablesVisited`). La variabile rimanente corrisponde al complemento e quindi vengono trovati tutti i predicati contenenti la variabile (`findOccurencies()`).
Dalla lista di modificatori del verbo vengono rimossi i predicati corrispondenti al complemento.

Dopo aver determinato gli argomenti dell'espressione logica, il sentence plan viene creato con la funzione `createPlan()`, la quale crea un dizionario contente gli argomenti (chiavi *subj*, *verb*, *obj*, *compl*) e, se presenti i rispettivi modificatori.

La lessicalizzazione dei predicati avviene grazie alla funzione `translatePlan()`, la quale prende in ingresso il sentence plan (`plan`) e il dizionario contenente le traduzioni (`lex`). Il file contenente le traduzioni (*eng_ita_lex.csv*) ha una struttura del tipo

$$term\_eng; term\_ita$$

e viene trasformato in un dizionario (attraverso la funzione `loadLexicon()`) che ha come chiavi i termini inglesi e come valori la corrispondente traduzione italiana.
La funzione `translatePlan()` cicla su tutte le chiavi di `plan` e, nel caso in cui il predicato non risulti vuoto, vengono valutate le corrispondenti chiavi. In particolare, nel caso in cui il predicato sia una lista (come nel caso del complemento), viene effettuata la traduzione di tutti i nomi dei vari predicati. Inoltre, nel caso in cui siano presenti dei modificatori, identificati dalla chiave *mod*, vengono tradotti anch'essi.
Dopo aver effettuato la traduzione dei nomi dei predicati degli argomenti, il dizionario viene salvato in formato JSON con la funzione `savePlanToJSON()`, in modo tale da poter essere utilizzato per creare le frasi in italiano con il realizer.

5. Realizzazione della frase

La realizzazione viene fatta in Java, utilizzando la libreria SimpleNLG-it [2]. Inizialmente vengono letti i JSON contenenti i sentence plan delle frasi analizzate e successivamente viene generata la frase in italiano attraverso la funzione `getRealisedSentence()`. La funzione permette di generare la frase in tre step:

- Creazione di una frase nominale (`NPPhraseSpec`) per il soggetto, con eventuali modificatori;
- Creazione di un predicato verbale (`VPPhraseSpec`) per il verbo, con eventuali modificatori;
- Creazione di una frase nominale (`NPPhraseSpec`) per l'oggetto, con eventuali modificatori;
- Aggiunta della parte della frase relativa ad un complemento.

5.1 Creazione frase nominale per il soggetto

Inizialmente viene creata una frase nominale attraverso la funzione `createNounPhrase()` in cui il nome corrisponde al nome del predicato del soggetto. Siccome nella struttura del sentence plan sono presenti le informazioni sintattiche dei predicati, è possibile impostare il numero e il genere

```
if (subjPred.get("num").equals("pl")) subject.setPlural(true);
if (subjPred.containsKey("gen")) {
    if (subjPred.get("gen").equals("f")) {
        subject.setFeature(LexicalFeature.GENDER, Gender.FEMININE);
}}
```

Inoltre, nel caso in cui il soggetto abbia dei modificatori, questi vengono aggiunti con la funzione `addModifier()` [5].

5.2 Creazione sintagma verbale per il verbo

Inizialmente viene creato un sintagma verbale con la funzione `createVerbPhrase()` in cui il verbo corrisponde al nome del predicato del verbo. Successivamente viene impostato il tempo verbale della frase in base alla chiave *tns* del predicato. Infine, vengono aggiunti i modificatori, se presenti.

```
private void setVerbTense(VPPhraseSpec clause, String tense) {
    if (tense.equals("pres")) {
        clause.setFeature(Feature.TENSE, Tense.PRESENT);
    } else if (tense.equals("ger")) {
        clause.setFeature(Feature.PROGRESSIVE, true);
        clause.setFeature(Feature.PERFECT, false);
    }
}
```

Avendo creato la frase nominale (`subject`) e la frase verbale (`verb`), è possibile creare un periodo con la funzione `createClause()`.

5.3 Creazione frase nominale per l'oggetto

Nel caso in cui il sentence plan contenga il predicato riferito all' oggetto, viene creata la frase nominale, viene impostata la coordinazione del numero (singolare o plurale) e vengono aggiunti i modificatori, se presenti. Infine nella frase costruita al passo precedente viene aggiunto il complemento oggetto attraverso la funzione `setObject()`.

5.4 Aggiunta di altri complementi

Nel caso in cui il sentence plan contenga il predicato riferito ad un complemento diverso dall'oggetto, tutti i nomi dei predicati corrispondenti vengono aggiunti come complemento, con la funzione `clause.addComplement()` [5].

Alla fine la frase completa viene restituita con la funzione `realiseSentence()`, la quale permette di combinare i vari argomenti e rendere il risultato finale sintatticamente e morfologicamente corretto.

6. Risultati

Di seguito i risultati ottenuti:

Frase inglese	**Frase italiano**
You are imagining things	Tu stai immaginando cose.
There is a price on my head	Prezzo esiste sopra mia testa.
Your big opportunity is flying out of here	Tua opportunità grande sta volando da qui via.
You are imagining my head	Tu stai immaginando testa mia.

Come si può notare è presente la coordinazione numerica e del genere. Inoltre, i tempi verbali sono rispettati. In alcuni casi l'ordine delle parole non risulta corretto (*da qui via*).
Da una successiva analisi, il problema riscontrato nell'ordine delle parole sembra derivare da una scelta progettuale secondo la quale la parola *out* (*via*) risulta essere un modificatore del verbo e per questo motivo viene aggiunta al termine della frase in quanto si utilizza il metodo `addModifier()`.

Eventuali miglioramenti potrebbero risolvere gli errori appena descritti e integrare le seguenti funzionalità:

- Aggiunta dei quantificatori e gestione degli articoli;
- Migliorare la gestione dei modificatori;
- Espandere l'utilizzo del sistema per casi d'uso diversi da quelli richiesti.

SITOGRAFIA

[1] https://github.com/simplenlg/simplenlg
[2] https://github.com/alexmazzei/SimpleNLG-IT/blob/master/docs/Testsimplenlgit.java
[2] https://www.nltk.org/
[3] https://github.com/nltk/nltk_teach/blob/master/examples/grammars/book_grammars/simple-sem.fcfg
[4] http://www.nltk.org/book/ch10.html
[5] http://www.ling.helsinki.fi/kit/2008s/clt310gen/docs/simplenlg-v37.pdf