

- Tipe SBJ menunjukan peran kata I sebagai subject terhadap kata want, Tipe OBJ menunjukan peran kata elephant sebagai oject dari kata shot, tipe DETMOD menunjukan peran kata my dan an

untuk kepemilikan/kepemilikan, tipe PMOD menunjukkan peran kata pajamas sebagai keterangan dan tipe NMOD menunjukkan peran kata “in” sebagai pengubah kata benda

#### - Tipe Relasi Dependency Tree dep\_grammar\_2

-Tipe SBJ menunjukkan peran kata man sebagai subject terhadap kata taught peran kata dog sebagai subject terhadap kata play, tipe DETMOD menunjukkan peran kata his menerangkan kepemilikan/kepemilikan, dan tipe NMOD menunjukkan peran kata dog, golf, dan to sebagai pengubah kata benda

### 3. Perbedaan Projective parser dan Non-Projective parser

Projective parser ketika semua kata ditulis dalam urutan linear yang sesuai dengan tata bahasa struktur frase sedangkan Non-Projective parser tata bahasa struktur frase bersifat lebih fleksibel.

#### 4. Format CONLL

Adalah sebuah format yang mewakili korpus dengan satu kata per baris, setiap baris kata berisi 10 kolom yang dipisahkan tab dengan informasi tentang kata tersebut. Atributnya terdiri dari ID yaitu indeks kata, FORM yaitu bentuk kata, LEMMA yaitu bentuk kata yang disederhanakan, CPOSTAG yaitu POST-tag terperinci, FEATS yaitu kumpulan fitur yang diabaikan secara default, HEAD yaitu kepala token saat ini, DEPREL yaitu relasi ketergantungan, PHEAD yaitu kepala proyektif, PDEPREL yaitu relasi ketergantungan untuk PHEAD

## Bagian 2

### 5. Dependency Grammar untuk parsing Projective Parser

#### 1. Definiskan Dependency Parser

```
[106] dep_grammar_1 = nltk.DependencyGrammar.fromstring("""
    'melihat' -> 'aziz' | 'seorang' | 'dengan'
    'seorang' -> 'penjahat' | 'dengan'
    'dengan' -> 'pistol'
    'pistol' -> 'miliknya'
    """)
```

Alasan grammar tersebut sesuai karena berdasarkan penulisan kata ditulis dalam urutan linear yang sesuai dengan tata bahasa. Kegunaan Dependency grammar disini untuk mendefinisikan kata mana saja yang akan dilakukan parsing

#### 2. Print Dependency Grammar

```
Dependency grammar with 7 productions
'melihat' -> 'aziz'
'melihat' -> 'seorang'
'melihat' -> 'dengan'
'seorang' -> 'penjahat'
'seorang' -> 'dengan'
'dengan' -> 'pistol'
'pistol' -> 'miliknya'
```

Ini berguna untuk melihat head(kepala) dan juga dependentnya (yang bergantung dengan head)

### 3. Lakukan parsing dengan Parser Dependency Projective

```
pdp = nltk.ProjectiveDependencyParser(dep_grammar_1)
sent = 'aziz melihat seorang penjahat dengan pistol miliknya'.split()
trees = pdp.parse(sent)

for tree in trees:
    print(tree)

(melihat aziz (seorang penjahat) (dengan (pistol miliknya)))
(melihat aziz (seorang penjahat (dengan (pistol miliknya))))
```

Berdasarkan hasil parsing diatas sebuah kata dan semua turunannya membentuk urutan kata yang berdekatan dalam kalimat . Struktur ketergantungan dalam tanda kurung ini juga dapat ditampilkan sebagai pohon, di mana tanggungan ditampilkan sebagai dependect dari headnya.

- Headnya yaitu kata “melihat” dan dependentnya yaitu “aziz”, “seorang”, dan “dengan”
- Headnya yaitu kata “seorang” dan dependentnya yaitu “penjahat”, ”dengan” dan “pistol”
- Headnya yaitu kata “dengan” dan dependentnya yaitu ‘pistol”
- Headnya yaitu kata “pistol” dan dependentnya yaitu ”miliknya”

### 5. Dependency Grammer untuk parsing Non-Projective Parser

#### 1. Definisikan Dependency Parser untuk Parser NON-Projective

```
dep_grammar_2 = nltk.DependencyGrammar.fromstring("""
    'mengajak' -> 'bermain' | 'wanita'
    'wanita' -> 'itu'
    'bermain' -> 'basket' | 'teman' | 'untuk'
    'teman' -> 'kecilnya'
    """)
```

Alasan grammar tersebut tata bahasa struktur frase bersifat lebih fleksibel dan ditulis tidak dengan tata bahasa . Kegunaan Dependency grammar disini untuk mendefinisikan kata mana saja yang akan dilakukan parsing

#### 2. Print Dependency Grammar

```
print(dep_grammar_2)

Dependency grammar with 7 productions
'mengajak' -> 'bermain'
'mengajak' -> 'wanita'
'wanita' -> 'itu'
'bermain' -> 'basket'
'bermain' -> 'teman'
'bermain' -> 'untuk'
'teman' -> 'kecilnya'
```

Ini berguna untuk melihat head(kepala) dan juga dependentnya (yang bergantung dengan head)

### 3. Lakukan parsing dengan Parser Non Dependency Projective

```

3: {'address': 3,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [1, 4, 8]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'mengajak'},
4: {'address': 4,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [7, 9]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'bermain'},
5: {'address': 5,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [6]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'teman'},
6: {'address': 6,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,

```

```

4: {'address': 4,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'bermain'},
5: {'address': 5,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [6]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'teman'},
6: {'address': 6,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'kecilnya'},
7: {'address': 7,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,

```

```

0: {'address': 0,
   'ctag': 'TOP',
   'deps': defaultdict(<class 'list'>, {'ROOT': [3]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': 'TOP',
   'word': None},
1: {'address': 1,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [2]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'wanita'},
2: {'address': 2,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'itu'},
3: {'address': 3,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [1, 4, 8]}),
   'feats': None,
   'head': None,
   'lemma': None,

```

```

9: {'address': 9,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'basket'}}
(<function DependencyGraph.__init___.<locals>.<lambda> at 0x7f9e2d271e0>,
0: {'address': 0,
   'ctag': 'TOP',
   'deps': defaultdict(<class 'list'>, {'ROOT': [3]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': 'TOP',
   'word': None},
1: {'address': 1,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {'': [2]}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,
   'tag': None,
   'word': 'wanita'},
2: {'address': 2,
   'ctag': None,
   'deps': defaultdict(<class 'list'>, {}),
   'feats': None,
   'head': None,
   'lemma': None,
   'rel': None,

```

Berdasarkan hasil parsing diatas hasil parsing berupa dictonery berisi key dan value, dimana key berisi address, ctag, deps, feats, head, lemma, rel, tag dan word. Kita bisa mengakses data melalui address dan juga terdapat informasi root dari parsing tersebut. Hasil dari **Non Dependency Projective** membentuk banyak parses karena struktur frase bersifat lebih fleksibel dan ditulis tidak dengan tata bahasa