University of Malta
BSc IT (Hons.) Year III

CSA305: Natural Language Processing

Lecture 7

Definite Clause Grammars Part II

- Handling Agreement
- Building Syntactic Structure
- Handling Subcategorisation

Department of Computer Science and AI
2000/01

Using Features: Agreement

Aim: INCLUDE

The dog chases the cats.
The dogs chase the cats.
Lina rat kelb.
John is English and his father is Maltese

Aim: EXLUDE

The dog chase the cats.*
The dogs chases the cats.*
Lina ra kelb*.
John is English and her father is Maltese*

- Agreement phenomena concern number, person and gender.
- The grammar we have seen so far cannot handle agreement phenomena.

How Not to Handle Agreement

The wrong way to handle agreement is to "multiply categories", since this greatly increases the number of grammar rules.

```
s      --> ssing.
s      --> splur.

ssing  -->  npsing vpsing.
ssplur -->  npplur vpplur.

npsing -->  detsingfem   nsingfem
npsing -->  detsingmasc  nsingmasc


npplur -->  detplurfem  nplurfem
npplur -->  detplurmasc npplurmasc
```

Things get even worse when person agreement is added.

Adding Agreement Features

Basic idea: add argument positions to handle agreement features.

Before

```
s --> np, vp.
```

After: this effect

```
s --> np(sing), vp(sing).
s --> np(plur), vp(plur).
```

can be achieved with

```
s --> np(NUM), vp(NUM).
```

To get agreement this rule to work correctly we must bear in mind that information about number (and other features) derives from the lexicon.

Adding Agreement Information to the Lexicon

- Lexical Entry

```
lex(dog,n,sing).
lex(dogs,n,plur).
```

- Lexical Category Relation

```
noun(X,AGR) :- lex(X,n,AGR).
```

- Lexical Category Rule

```
n(AGR) --> [X], { noun(X,AGR). }
```

- Phrasal Rule

```
np(AGR) --> n(AGR).
```


Complete Grammar for Agreement

```
s       --> np(A), vp(A).
np(A)   --> n(A).
np(A)   --> d(A), n(A).
vp(A)   --> v(A).
vp(A)   --> v(A), np(A1).
vp(A)   --> v(A), np(A1), np(A2).


n(A) --> [X], { noun(X,A) }.
d(A) --> [X], { det(X,A) }.
v(A) --> [X], { verb(X,A) }.


noun(X,A) :- lex(X,noun,A).
verb(X,A) :- lex(X,verb,A).
det(X,A)  :- lex(X,det,A).
prep(X)   :- lex(X,prep).
```


Adding Gender Information: 2 Approaches

1. Add a second argument to the whole grammar, so that a simple NP rule which looked like

   ```
   np(NUM) --> d(NUM), n(NUM).
   ```

   would now look like:

   ```
   np(NUM,GEN) --> d(NUM,GEN), n(NUM,GEN).
   ```

   This approach works fine but (a) the grammar has to be changed throughout and (b) its readibility is slightly decreased.

2. Retain the single extra argment, but change it from an atomic symbol to a structure such as `agr(Number, Gender)` . This would yield the rule:

   ```
   np(agr(NUM,GEN)) --> d(agr(NUM,GEN)),
                        n(agr(NUM,GEN)).
   ```


Adding Gender Agreement

```
np(agr(NUM,GEN)) --> np(agr(NUM,GEN)),
                     vp(agr(NUM,GEN)).
```

which in isolation is of course equivalent to

```
np(A) --> d(A), n(A).
```

i.e. the original grammar rule.

So approach (2) requires little, if any, change to the grammar which is a great advantage. However, the lexicon must change, as must the interface to it.

Adding Further Agreement Information to the Lexicon

- Lexical Entry

  This must include extra agreement information.

  `lex(man,n,sing,masc).  lex(men,n,plur,masc)`

- Lexical Category Relation

  This must be adapted to build the agreement structure

  `noun(X,agr(NUM,GEN)) :- lex(X,n,NUM,GEN).`

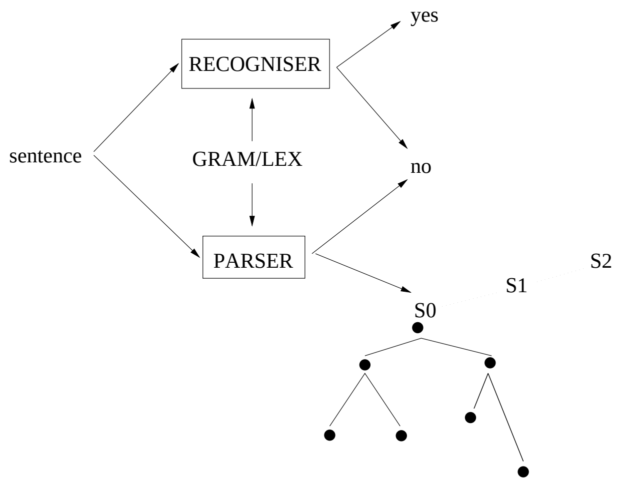
- Lexical Category Rule is unchanged

  `n(AGR) --> [X], noun(X,AGR).`

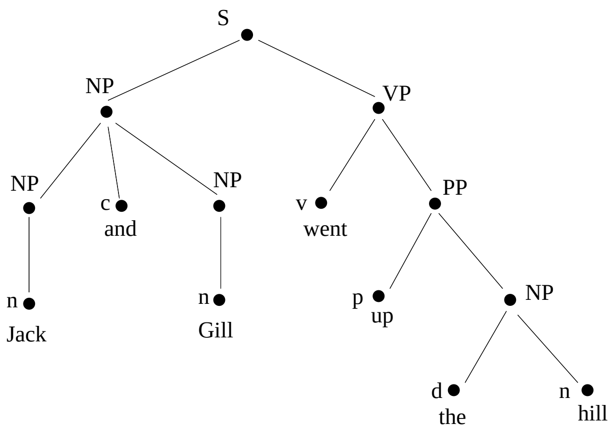
- Phrasal Rule is unchanged

  `np(AGR) --> n(AGR).`

Parsing versus Recognition



Prolog Terms are Tree Structures




```
s(np(np(n('Jack')),
     c(and),
     np(n('Gill'))),
  vp(v(went),pp(p(up),np(d(the),n(hill)))))
```


Deriving Structure: Phrasal Category Rules

Basic idea is to give grammar rules an extra argument place for holding structure. Note that in this set of examples agreement information is supressed for the sake of readability.

Before

```
s --> np, vp.
```

After

```
s(s(Np,Vp)) --> np(Np), vp(Vp).
```

Deriving Structure: Lexical Category Rules

The argument-less lexical category is given an argument standing for the syntactic structure associated with lexical categories.

Before

```
n --> [X], { noun(X) }.
```

After

```
n(Y) --> [X], { noun(X,Y) }
```

The lexical predicate `noun(X)`, where X is a word, has become a 2-place relation `noun(X,Y)` where `Y` is the associated syntax tree.

Deriving Structure: The Lexicon

The unary lexical predicate now associates the syntactic structure.

Before

```
noun(X) :- lex(X,noun).
```

After

```
noun(X,n(X)) :- lex(X,noun).
```


Complete Grammar for Assigning Structure

```
s(s(Np,Vp))          -->  np(Np), vp(Vp).
np(np(N))            -->  n(N).
np(np(D,N))          -->  d(D), n(N).
vp(vp(V))            -->  v(V).
vp(vp(V,Np))         -->  v(v), np(Np).
vp(vp(V,Np1,Np2))    -->  v(V), np(Np1),
                          np(Np2).

n(Y)  -->  [X], { noun(X,Y) }.
d(Y)  -->  [X], { det(X,Y) }.
v(Y)  -->  [X], { verb(X,Y) }.

noun(X,n(X))  :-  lex(X,noun).
verb(X,v(X))  :-  lex(X,verb).
det(X,d(X))   :-  lex(X,det).

lex(the,det).
lex(cat,noun).
lex('John',noun).
lex(ate,verb).
```


Results

```
2 ?-  s(X,['John',ate,the,cat],[]).
X = s(np(n('John')), vp(v(ate), np(d(the), n(cat)))) ;

?- np(X,[the,cat],[]).
   Call:  (  7) np(_G215, [the, cat], []) ? creep
   Call:  (  8) n(_G318, [the, cat], []) ? creep
   Call:  (  9) noun(the, _G318) ? creep
   Call:  ( 10) lex(the, noun) ? creep
   Fail:  ( 10) lex(the, noun) ? creep
   Fail:  (  9) noun(the, _G318) ? creep
   Fail:  (  8) n(_G318, [the, cat], []) ? creep
   Redo:  (  7) np(_G215, [the, cat], []) ? creep
   Call:  (  8) d(_G318, [the, cat], _L129) ? creep
   Call:  (  9) det(the, _G318) ? creep
   Call:  ( 10) lex(the, det) ? creep
   Exit:  ( 10) lex(the, det) ? creep
   Exit:  (  9) det(the, d(the)) ? creep
   Call:  (  9) _L129 = [cat] ? creep
   Exit:  (  9) [cat] = [cat] ? creep
   Exit:  (  8) d(d(the), [the, cat], [cat]) ? creep
   Call:  (  8) n(_G319, [cat], []) ? creep
   Call:  (  9) noun(cat, _G319) ? creep
   Call:  ( 10) lex(cat, noun) ? creep
   Exit:  ( 10) lex(cat, noun) ? creep
   Exit:  (  9) noun(cat, n(cat)) ? creep
   Call:  (  9) [] = [] ? creep
   Exit:  (  9) [] = [] ? creep
   Exit:  (  8) n(n(cat), [cat], []) ? creep
   Exit:  (  7) np(np(d(the), n(cat)), [the, cat], []) ? cre

X = np(d(the), n(cat))
```

Dealing with Subcategorisation

```
vp(vp(V,Np),A)      --> v(V,A).
vp(vp(V,Np),A)      --> v(V,A), np(Np,_).
vp(vp(V,Np1,Np2),A) --> v(V,A), np(Np1,_), np(Np2,_).
```

The above rules cannot rule out such sentences as

1. The cat disappeared the dog
2. John met Mary the book
3. John discovered

i.e. sentences in which there is something wrong with the list of noun phrases following the verb. In (1) "disappear" cannot take an object. In (2), "meet" cannot be followed by 2 noun phrases. In (3), "discovered" requires an object.

The method we use to augment the grammar follows the same pattern as before, namely:

- Augment lexical entries with information necessary to discriminate different cases.
- Modify grammar to use that information

Subcategorisation - Approach I

In this approach we invent a label for each verb class pattern, and include this in the lexicon.

- Lexical entries now look like this

```
lex(sleep,v,...,intransitive,...).
lex(meet,v,...,transitive,...).
lex(give,v,...,ditransitive,...).
```

- Modification of the lexicon interface

  verb(A,TYPE) :- lex(_,v,...,TYPE,...).

- Modification of the Grammar

```
vp(A)     --> v(A,intransitive).
vp(A)     --> v(A,transitive), np(A1).
vp(A)     --> v(A,ditransitive), np(A1), np(A2).
v(A,TYPE) --> verb(A,TYPE).
```


Other Verb Complements

Theproblem with this approach is that there are more than three different verb patterns.

- Other Verb Complements
  - **np pp** *John told Bill about the accident*
  - **np s**
    John persuaded Bill to come
  - **np pp pp**
    John rented an apartment to Bill for Lm100
- Taking prepositions into account, about 40 different patterns are required to handle verbs in English.
- This will lead to an undesirable proliferation of grammar rules.

Subcategorisation - Approach 2

- Shift responsibility towards the lexicon The verb will then have a **subcat list**, i.e. a list of complement types in its lexical entry. Schematically, lexical entries might include the following information:
  - intransitive: lex(sleep,v,[],...).
  - transitive: lex(meet,v,[np],...).
  - ditransitive: lex(give,v,[np,np],...).
  - dative: lex(give,v, [np,pp]),...).
- Instead of many grammar rules, Have a **single** rule which allows a verb phrase to be formed from a verb followed by a sequence of zero or more "complements" where a complement is either a noun phrase, a prepositional phrase or a sentence.

Grammar Rule with Subcat List

Basic idea is to avoid multiple grammar rules of the form

```
vp  --> v([]).
vp  --> v([np]), np(...).
vp  --> v([np,np]), np(...), np(...).
vp  --> v([np,pp]), np(...), pp(...).
vp  --> v([np,s]), np(...), s(...).
```

with a single rule of the following type:

```
vp --> v(COMPLIST), comps(COMPLIST).
```

The comps rule is then defined separately to handle the multi-element list of complements.

```
comps([])    --> [].
comps([X|R]) --> x(X), comps(R).
```


Grammar dcgsubcat.pl

```
%% phrasal category rules

s(A)          --> np(A), vp(A).
np(A)         --> n(A).
np(A)         --> d(A), n(A).
vp(A)         --> v(A,SC), comps(SC).
comps([])     --> [].
comps([X|R])  --> x(X), comps(R).
x(np)         --> np(A).
x(vp)         --> vp(A).

% lexical category rules


n(A)    --> [X], { noun(X,A) }.
d(A)    --> [X], { det(X,A) }.
v(A,SC) --> [X], { verb(X,A,SC) }.
```


Grammar dcgsubcat.pl

```
% lexical category predicates

noun(X,agr(NUM,GEN))    :- lex(X,noun,NUM,GEN).
verb(X,agr(NUM,GEN),SC) :- lex(X,verb,NUM,GEN,SC).
det(X,agr(NUM,GEN))     :- lex(X,det,NUM,GEN).


% lexicon
% note the use of "_" to
% indicate a don't care value

lex(the,det,_,_).
lex(this,det,sing,_).
lex(these,det,plur,_).
lex(cat,noun,sing,_).
lex(john,noun,sing,masc).
lex(uther,noun,sing,masc).
lex(cornwall,noun,sing,masc).
lex(ate,verb,_,_).
lex(slept,verb,_,_,[]).
lex(stormed,verb,_,_,[np]).
lex(gave,verb,_,_,[np,np]).
lex(promised,verb,_,_,[np,vp]).
```


Assignment

- Design a DCG-like grammar formalism and parsing algorithm for left-recursive rules. [5 hours (30%)]
- Understanding syntactic properties of texts. [2 hours (15%)]
- Write a grammar/lexicon which covers all the sentences, and give a few examples of respective parse trees. [5 hours (30%)]
- Output a list of head nouns for all the noun phrases encountered in the texts ordered by frequency of occurrence. [3 hours (25%)].
- Full description on website.