Artificial Intelligence (CSBP480)

Introduction to
Natural Language Processing (3)

Use Extra Arguments

Lecture 8: More DCGs

- Theory
  - Examine two important capabilities offered by DCG notation:
    - Extra arguments
    - Extra tests
  - Discuss the status and limitations of DCGs

Extra arguments

- In the previous lecture we introduced basic DCG notation
- But DCGs offer more than we have seen so far
  - DCGs allow us to specify **extra arguments**
  - These extra arguments can be used for many purposes

Extending the grammar

- This is a simple grammar
- Suppose we also want to deal with sentences containing pronouns such as

  she chased him

  and

  he chased her
- What do we need to do?

```
s --> np, vp.
np --> det, n.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
```

Extending the grammar

- Add rules for pronouns
- Add a rule saying that noun phrases can be pronouns

- Is this new DCG any good?
- What is the problem?

```
s --> np, vp.
np --> det, n.
np --> pro.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
```

Some examples of grammatical strings accepted by this DCG

```
?- s([the, boy, chased, her], [ ]).
yes
?- s([the, cat, chased, him], [ ]).
yes
```

```
s --> np, vp.
np --> det, n.
np --> pro.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
```

Some examples of ungrammatical strings accepted by this DCG

```
?- s([the, cat, chased, he],[ ]).
yes
?- s([her, chased, a, boy], [ ]).
yes
s([her, chased, she], [ ]).
yes
```

```
s --> np, vp.
np --> det, n.
np --> pro.
vp --> v, np.
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro --> [he].
pro --> [she].
pro --> [him].
pro --> [her].
```

What is going wrong?

- The DCG ignores some basic facts about English
  - *she* and *he* are subject pronouns and cannot be used in the object position
  - *her* and *him* are object pronouns and cannot be used in the subject position
- It is obvious what we need to do: extend the DCG with information about subject and object
- How do we do this?

A naïve way: change notation…

```
s --> np_subject, vp.
np_subject --> det, n.
np_subject --> pro_subject.
np_object --> det, n.
np_object --> pro_object.
vp --> v, np_object.
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro_subject --> [he].
pro_subject --> [she].
pro_object --> [him].
pro_object --> [her].
```

Better way: use extra arguments

```
s --> np(subject), vp.
np(_) --> det, n.
np(X) --> pro(X).
vp --> v, np(object).
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
```

This works…

```
s --> np(subject), vp.
np(_) --> det, n.
np(X) --> pro(X).
vp --> v, np(object).
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
```

```
?- s([she, chased,him],[ ]).
yes
?- s([she, chased, he],[ ]).
no
?-
```

What is really going on?

- Recall that the rule:

  s --> np, vp.

  is really a syntactic sugar for:

  s(A,B):- np(A,C), vp(C,B).

What is really going on?

- Recall that the rule:

  s --> np,vp.

  is really syntactic sugar for:

  s(A,B):- np(A,C), vp(C,B).

- Then the rule

  s --> np(subject), vp.

  Is represented in prolog as:

  s(A,B):- np(subject,A,C), vp(C,B).

Listing noun phrases

```
s --> np(subject), vp.
np(_) --> det, n.
np(X) --> pro(X).
vp --> v, np(object).
vp --> v.
det --> [the].
det --> [a].
n --> [boy].
n --> [cat].
v --> [chased].
pro(subject) --> [he].
pro(subject) --> [she].
pro(object) --> [him].
pro(object) --> [her].
```

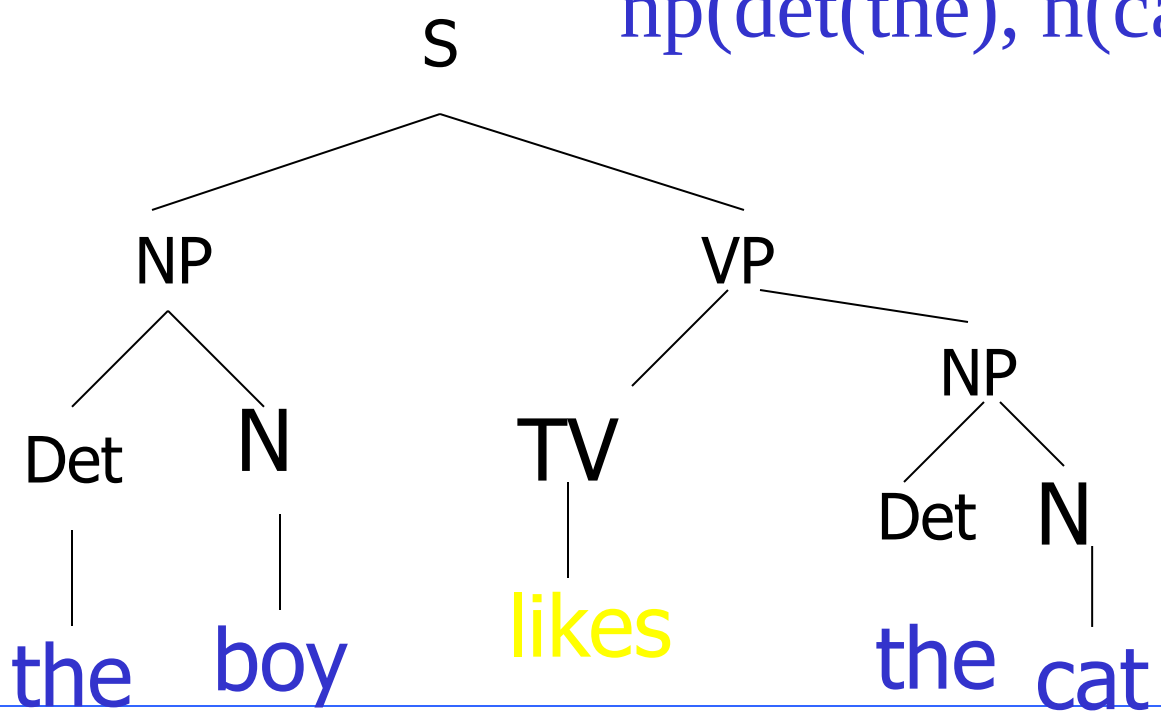
```
?- np(Type, NP, [ ]).
Type =_
NP = [the,boy];

Type =_
NP = [the,cat];

Type =_
NP = [a, boy];

Type =_
NP = [a, cat];

Type =subject
NP = [he]
```

Building parse trees

- The programs we have discussed so far have been able to recognise grammatical structure of sentences
- But we would also like to have a program that gives us an analysis of their structure
- In particular we would like to see the trees the grammar assigns to sentences

Syntax Tree

- s(np(det(the), n(boy)), vp(tv(likes), np(det(the), n(cat))))



Grammar rules in Prolog

- Prolog allows us to add arguments to the grammar rule.
- So, we can write the rules as:

```
s(s(NP,VP))-->  np(NP), vp(VP).
np(np(N)) -->  pronoun(N).
np(np(Det,N)) -->  det(Det), n(N).
vp(vp(V)) -->  itv(V).
vp(vp(TV,NP))-->  tv(TV), np(NP).
```

Grammar rules in Prolog

- We can use this lexicon (dictionary)

```
pronoun(pn(i))-->[i].
det(det(the))-->[the].
n(n(cat))-->[cat].
n(n(boy))-->[boy].
n(n(ball))-->[ball].
tv(tv(hit))-->[hit].
tv(tv(likes))-->[likes].
itv(itv(run))-->[run].
```

Reading from the keyborad

- Prolog has a built in predicate called readln(S).
- It allows you to read a line and put it in a list.
- We can use it to read a sentence:
  - **run :-**

```
readln(S),
s(Tree,S,[]),
write("Syntax Tree: "),
write(Tree).
```

Try the grammar

- Now load your grammar and run it.

```
| ?-run.
```