

A DCG for English using gap threading for unbounded dependencies

Detmar Meurers: Intro to Computational Linguistics I
OSU, LING 684.01

Towards a basic DCG for English: X-bar Theory

Generalizing over possible phrase structure rules, one can attempt to specify DCG rules fitting the following general pattern:

$X^2 \rightarrow \text{specifier}^2 X^1$
 $X^1 \rightarrow X^1 \text{ modifier}^2$
 $X^1 \rightarrow \text{modifier}^2 X^1$
 $X^1 \rightarrow X^0 \text{ complement}^{2*}$

To turn this general X-bar pattern into actual DCG rules,

- X has to be replaced by one of the atoms encoding syntactic categories, and
- the bar-level needs to be encoded as an argument of each predicate encoding a syntactic category.

2

Noun, preposition, and adjective phrases

```
n(2,Num) --> pronoun(Num).
n(2,Num) --> proper_noun(Num).
n(2,Num) --> det(Num), n(1,Num).
n(2,plur) --> n(1,plur).
n(1,Num) --> pre_mod, n(1,Num).
n(1,Num) --> n(1,Num), post_mod.
n(1,Num) --> n(0,Num).
...

p(2,Pform) --> p(1,Pform).
p(1,Pform) --> adv, p(1,Pform). % slowly past the window
p(1,Pform) --> p(0,Pform), n(2,_).
...

a(2) --> deg, a(1). % very simple
a(1) --> adv, a(1). % commonly used
a(1) --> a(0).
```

3

Verb phrases and sentences

```
v(2,Vform,Num) --> v(1,Vform,Num).
v(1,Vform,Num) --> adv, v(1,Vform,Num).
v(1,Vform,Num) --> v(1,Vform,Num), verb_postmods.
v(1,Vform,Num) --> v(0,intrans,Vform,Num).
v(1,Vform,Num) --> v(0,trans,Vform,Num), n(2).
v(1,Vform,Num) --> v(0,ditrans,Vform,Num), n(2), n(2).
...

s(Vform) --> n(2,Num), v(2,Vform,Num).
```

4

From local to non-local dependencies

- A head generally realizes its arguments locally within its head domain, i.e., within a local tree if the X-bar schema is assumed.
- Certain kind of constructions resist this generalization, such as, for example, the *wh*-questions discussed below.
- How can the non-local relation between a head and such arguments be licensed? How can the properties be captured?

5

A first example: *Wh*-elements

Wh-elements can have different functions:

- (1) a. Who did Hobbs see _ ? Object of verb
b. Who do you think _ saw the man? Subject of verb
c. Who did Hobbs give the book to _ ? Object of prep
d. Who did Hobbs consider _ to be a fool? Object of obj-control verb

Wh-elements can also occur in subordinate clauses:

- (2) a. I asked who the man saw _ .
b. I asked who the man considered _ to be a fool .
c. I asked who Hobbs gave the book to _ .
d. I asked who you thought _ saw Hobbs.

6

Different categories can be extracted:

- (3) a. Which man did you talk to _ ? NP
b. [To [which man]] did you talk _ ? PP
c. [How ill] has the man been _ ? AdjP
d. [How frequently] did you see the man _ ? AdvP

This sometimes provides multiple options for a constituent:

- (4) a. Who does he rely [on _]?
b. [On whom] does he rely _ ?

Unboundedness:

- (5) a. Who do you think Hobbs saw _ ?
b. Who do you think Hobbs said he saw _ ?
c. Who do you think Hobbs said he imagined that he saw _ ?

7

Unbounded dependency constructions

An unbounded dependency construction

- involves constituents with different functions
- involves constituents of different categories
- is in principle unbounded

Two kind of unbounded dependency constructions (UDCs)

- Strong UDCs
- Weak UDCs (*easy*, purpose infinitives, ...) → not addressed here

8

Strong UDCs

An overt constituent occurs in a non-argument position:

Topicalization:

(6) Kim_i, Sandy loves _{-i}.

Wh-questions:

(7) I wonder [who_i Sandy loves _{-i}].

Wh-relative clauses:

(8) This is the politician [who_i Sandy loves _{-i}].

It-clefts:

(9) It is Kim_i [who_i Sandy loves _{-i}].

Pseudoclefts:

(10) [What_i Sandy loves _{-i}] is Kim_i.

9

Link from filler to gap needed to identify category

(11) a. Kim_i, Sandy trusts _{-i}.

b. [On Kim]_i, Sandy depends _{-i}.

(12) a. * [On Kim]_i, Sandy trusts _{-i}.

b. * Kim_i, Sandy depends _{-i}.

And this link has to be established for an unbounded length:

(13) a. Kim_i, Chris knows Sandy trusts _{-i}.

b. [On Kim]_i, Chris knows Sandy depends _{-i}.

(14) a. * [On Kim]_i, Chris knows Sandy trusts _{-i}.

b. * Kim_i, Chris knows Sandy depends _{-i}.

(15) a. Kim_i, Dana believes Chris knows Sandy trusts _{-i}.

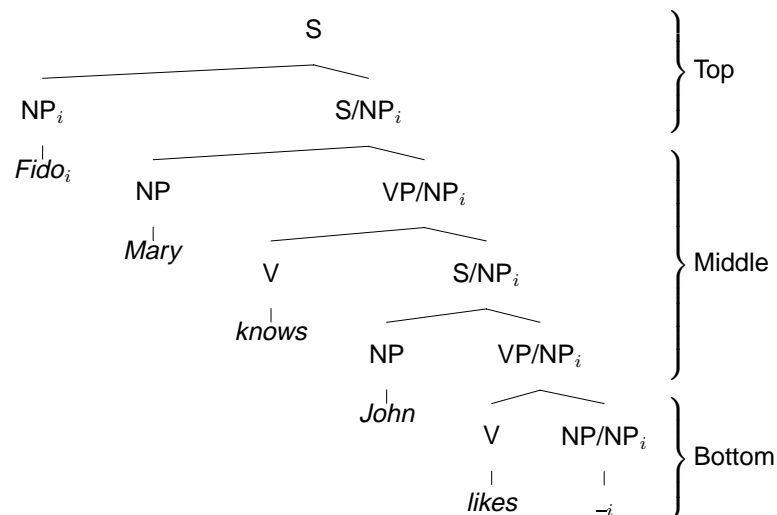
b. [On Kim]_i, Dana believes Chris knows Sandy depends _{-i}.

(16) a. * [On Kim]_i, Dana believes Chris knows Sandy trusts _{-i}.

b. * Kim_i, Dana believes Chris knows Sandy depends _{-i}.

10

An example for a strong UDC



11

A small DCG (dcg/udc/dcg_basis.pl)

```

np --> [mary]
      ; [john]
      ; [fido].

s --> np,
      vp.

vp --> vt,
      np.

p --> [to].
pp --> p,
      np.

vt --> [loves].
vd --> [gives].
vs --> [knows].

vp --> vs,
      s.
  
```

Towards a Prolog encoding of strong UDCs

12

A mini grammar with gaps (dcg/udc/dcg_gaps1.pl)

```
% 1) Top of UDC: realizing filler
s(nogap) --> np(nogap), s(gap).

% 2) Middle of UDC: passing info
s(GapInfo) --> np(nogap), vp(GapInfo). % no subject gaps
vp(GapInfo) --> vt,          np(GapInfo).

% 3) Bottom of UDC
np(gap) --> [].

% "Lexicon"
np(nogap) --> [mary];[john];[fido].

vt --> [loves].
```

Towards different kinds of gaps (dcg/udc/dcg_gaps2.pl)

```
% 1) Top of UDC: realizing filler
s(nogap) --> np(nogap), s(gap).

s(nogap) --> pp(nogap), s(gap).

% 2) Middle of UDC: passing info
s(GapInfo) --> np(nogap), vp(GapInfo). % no subject gaps

vp(GapInfo) --> vt, np(GapInfo).
vp(GapInfo) --> vd, np(GapInfo), pp(nogap).
vp(GapInfo) --> vd, np(nogap), pp(GapInfo).

pp(GapInfo) --> p, np(GapInfo).
```

```
% 3) Bottom of UDC
np(gap) --> [].
pp(gap) --> [].
```

```
% "Lexicon"
np(nogap) --> [mary];[john];[fido].
p --> [to].
vt --> [loves].
vd --> [gives].
```

Different kinds of gaps (dcg/udc/dcg_gaps3.pl)

```
% 1) Top of UDC: realizing filler
s(nogap) --> np(nogap), s(gap(np)).

s(nogap) --> pp(nogap), s(gap(pp)).

% 2) Middle of UDC: passing info
s(GapInfo) --> np(nogap), vp(GapInfo). % no subject gaps

vp(GapInfo) --> vt, np(GapInfo).
vp(GapInfo) --> vd, np(GapInfo), pp(nogap).
vp(GapInfo) --> vd, np(nogap), pp(GapInfo).

pp(GapInfo) --> p, np(GapInfo).
```

```
% 3) Bottom of UDC
np(gap(np)) --> [].
pp(gap(pp)) --> [].
```

```
% "Lexicon"
np(nogap) --> [mary];[john];[fido].
p --> [to].
vt --> [loves].
vd --> [gives].
```

From hardcoded gap percolation to gap threading

Two problems of current encoding:

- Two rules are needed to license ditransitive VPs.
- In sentences without topicalization, two identical analyses arise for ditransitive VPs.

Idea:

- Use difference-list encoding to thread occurrence of gaps through the tree ("gap threading").

An encoding using gap threading (dcg/udc/dcg_gaps4.pl)

```
% 1) Top of UDC: realizing filler
```

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

```
% 2) Middle of UDC: passing info
```

```
s(G0,G) --> np([],[]), vp(G0,G).
```

```
vp(G0,G) --> vt, np(G0,G).
vp(G0,G) --> vd, np(G0,G1), pp(G1,G).
pp(G0,G) --> p, np(G0,G).
```

```
% 3) Bottom of UDC
np([gap(np)],[]) --> [].
pp([gap(pp)],[]) --> [].
```

```
% "Lexicon"
np(X,X) --> [mary];[john];[fido].
p --> [to].
vt --> [loves].      vd --> [gives].
```

Reading assignment

Read the following chapters from the lecture notes:

- Chapter 4: *DCGs as a Grammar Formalism*
- Chapter 5: *Unbounded Dependencies in DCGs*