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?

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 object-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.*

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 _ ?*

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

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* [*who$_i$ Sandy loves $\__i$*].

Pseudoclefts:

(10) [*What Kim$_i$ loves $\__i$*] *is Sandy.*

Weak UDCs

No overt constituent in a non-argument position:

Purpose infinitive (*for-to* clauses):

(11) *I bought it$_i$ for Sandy to eat $\_{i}$.*

Tough movement:

(12) *Sandy$_i$ is hard to love $\_{i}$.*

Relative clause without overt relative pronoun:

(13) *This is* [*the politician*]$_i$ [*Sandy loves* $\_{i}$].

It-clefts without overt relative pronoun:

(14) *It is Kim$_i$* [*Sandy loves* $\_{i}$].

More on the link between filler and gap

Link between filler and gap with category information needed:

(15) a. *Kim$_i$, Sandy trusts $\_{i}$.*
b. [*On Kim*]$_i$, *Sandy depends $\_{i}$.*

(16) a. * [*On Kim*]$_i$, *Sandy trusts $\_{i}$.*
b. * *Kim$_i$, Sandy depends $\_{i}$.*

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

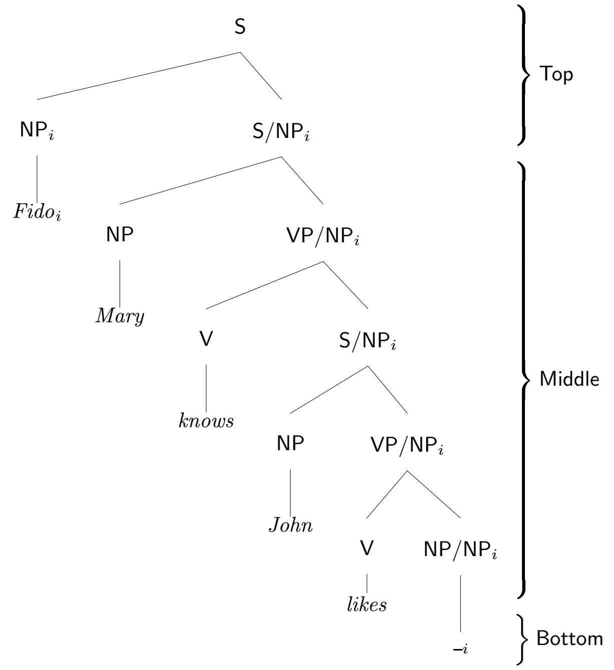
(17) a. *Kim$_i$, Chris knows Sandy trusts $\_{i}$.*
b. [*On Kim*]$_i$, *Chris knows Sandy depends $\_{i}$.*

(18) a. * [*On Kim*]$_i$, *Chris knows Sandy trusts $\_{i}$.*
b. * *Kim$_i$, Chris knows Sandy depends $\_{i}$.*

(19) a. *Kim$_i$, Dana believes Chris knows Sandy trusts $\_{i}$.*
b. [*On Kim*]$_i$, *Dana believes Chris knows Sandy depends $\_{i}$.*

(20) a. * [*On Kim*]$_i$, *Dana believes Chris knows Sandy trusts $\_{i}$.*
b. * *Kim$_i$, Dana believes Chris knows Sandy depends $\_{i}$.*

An example for a strong UDC



A small DCG to start from
(dcg_basis.pl)

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

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

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

s -->   np,
        vp.

vp -->  vt,
        np.

vp -->  vd,
        np,
        pp.

vp -->  vs,
        s.
```

Adding gaps to a reduced grammar
(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),    % no subject gaps
        vp(GapInfo).

vp(GapInfo) -->
        vt,
        np(GapInfo).

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

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

vt --> [loves].
```


Different kinds of gaps
(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),    % no subject gaps
               vp(GapInfo).

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].
```


Correcting treatment of different kinds of gaps
(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),    % no subject gaps
               vp(GapInfo).

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_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(GapIn,GapOut) --> np([],[]),    % no subject gaps
                    vp(GapIn,GapOut).

vp(GapIn,GapOut) --> vt,
                     np(GapIn,GapOut).

vp(GapIn,GapOut) --> vd,
                     np(GapIn,GapMid),
                     pp(GapMid,GapOut).

pp(GapIn,GapOut) --> p,
                     np(GapIn,GapOut).
```



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

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