Asymmetric C-command Sets

Meaghan Fowlie
mfowlie@ucla.edu

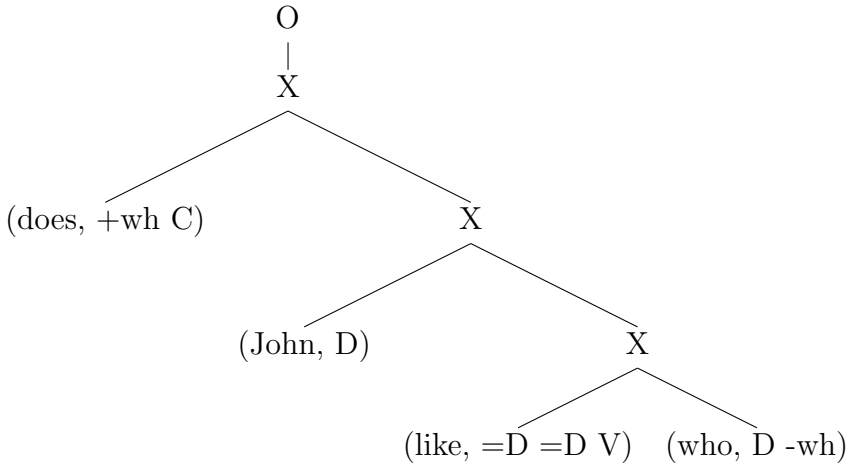
April 13
UCLA MathLing Circle

1 Terminology

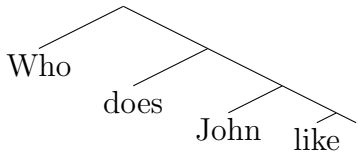
I'm working within Ed Stabler's minimalist grammars.

- ***Derivation tree***: structure that describes the functions used to derive the sentence. The relation represented by arcs is *takes as argument*
- ***Derived tree*** *(or Syntax tree)*: structure that describes the hierarchical organisation of the sentence derived. The partial order represented is the relation *dominates*.
- ***Address***: the address of a node in a tree is a string of 0s and 1s. The first number is 0 if it is down the left branch from the root and 1 if it is down the rght branch of the root. The second number is 0 if it is down the left branch from the second node and 1 if down the right, etc. See (3)

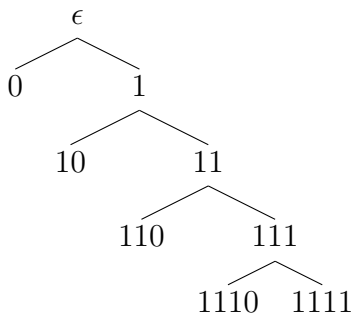
(1) Derivation tree:



(2) Derived (syntax) tree:



(3) Addressing:



A derived tree is uniquely determined by a derivation tree (if the Shortest Move Contraint is observed).

2 C-command

Definition A Dom B means A dominates B, where *dominates* is the partial order on the nodes that defines a syntax tree

Definition A *immediately dominates* B (A ImmDom B) iff A Dom B and $\nexists C$ such that A Dom B, C Dom B, (and C $\neq$ A or B).[1]

Definition C-command: A CC B iff $\exists C$ s.t. C ImmDom A and C Dom B.

Definition Asymmetric c-command: A ACC B iff (A CC B) but $\neg$(B CC A)

- Computing c-command (post-Move) for any two addresses in the derived (syntax) tree is easy

Proposition 2.1 *Suppose node A has address of length n and node B of length m. A c-commands B iff the first n-1 digits of the addresses match, the nth does not, and $n \leq m$. i.e. A CC B iff*
$Add(A) = d_1 d_2 ... d_{n-1} \mathbf{d_n}$
$Add(B) = d_1 d_2 ... d_{n-1} \mathbf{d'_n} (d_{n+1} ... d_m)$

Proposition 2.2 *Suppose node A has address of length n and node B of length m. A* ***asymmetrically*** *c-commands B iff the first n-1 digits of the addresses match, the nth does not, and* $\mathbf{n} < \mathbf{m}$*. i.e. A CC B iff*
$Add(A) = d_1 d_2 ... d_{n-1} \mathbf{d_n}$
$Add(B) = d_1 d_2 ... d_{n-1} \mathbf{d'_n} d_{n+1} ... d_m$

Example Node 110 asymmetrically c-commands node 1111 because all but the last digit of 110 is a prefix of 1111 (i.e. both start with 11). The next number does not match (0 vs 1) and 1111 is longer than 110.

[1] OR, ImmDom is the relation on the nodes that defines the syntax tree as a *graph*

- However, if you want the whole c-command set for the tree, with this method you have to check every pair of addresses (and find out what addresses are in your tree to begin with).
- **Question:** Can you get the c-command set of the derived tree directly from the derivation tree?

3 C-command sets

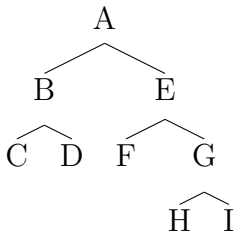
Definition *C-command set*: The binary relation CC on the nodes of tree T $ACC_T = \{\langle A, B\rangle |$ A c-commands B$\}$ is the *c-command set for T*.

Definition *Asymmetric C-command set*: The binary relation ACC on the nodes of tree T $ACC_T = \{\langle A, B\rangle |$ A asymmetrically c-commands B$\}$ is the *asymmetric c-command set for T*.

Research question: *What are the structures of the c-command sets of a tree?*

Suppose we have syntax (derived) tree (4)

(4)



The c-command set CC is:

(5) $\{\langle B, E\rangle, \langle E, B\rangle, \langle C, D\rangle, \langle D, C\rangle, \langle F, G\rangle, \langle G, F\rangle, \langle H, I\rangle, \langle I, H\rangle, \langle B, F\rangle, \langle B, G\rangle, \langle B, H\rangle, \langle B, I\rangle, \langle E, C\rangle, \langle E, D\rangle, \langle F, H\rangle, \langle F, I\rangle\}$

Notice that a c-command set is not transitive. For example, $\langle B, E\rangle, \langle E, C\rangle \in CC$ but $\langle B, C\rangle \notin CC$.

It is anti-symmetric (neither symmetric nor asymmetric)... I can't think of anything interesting to say about the structure of a normal c-command set. However!

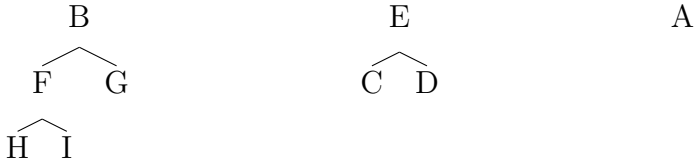
The *asymmetric* c-command relation on nodes ACC is the following set:

(6) $\{\langle B, F\rangle, \langle B, G\rangle, \langle B, H\rangle, \langle B, I\rangle, \langle E, C\rangle, \langle E, D\rangle, \langle F, H\rangle, \langle F, I\rangle\}$

The relation is in fact a **strict partial order:** it is irreflexive, asymmetric, and transitive. In fact, it can be represented by a set of irreflexive trees.

Here is the set of trees describing the poset, where the (directed) arc means "asymmetrically c-commands"[2]:

(7)



Proposition 3.1 *Asymmetric c-command (ACC) is a strict partial order with the chain condition.*

Proof

1. **irreflexive:** by definition of c-command
2. **asymmetric**: by definition of asymmetric c-command, of course
3. **transitive**: suppose A ACC B and B ACC D. Show A ACC D.

   A ACC B so $\exists C$ such that C ImmDom A and C Dom B.

   B ACC D so $\exists E$ such that E ImmDom B and E Dom D.

   Both C and E dominate B, and *dominate* is a rooted tree so either C dominates E or E dominates C or C = E.

   (a) Suppose C = E.

   Then C ImmDom A and C ImmDom B so A and B are sisters. Then ¬(A ACC B).

   (b) Suppose E dom C.

   Now, E Dom C and C Dom A, so E Dom A.

   So we have that E ImmDom B and E Dom A, so B ACC A, by definition of ACC. But A ACC B and ACC is asymmetric. Contradiction.

   (c) So C dom E. Now, C Dom E and E Dom D. By transitivity of Dom, C Dom D

   So we have that C ImmDom A and C Dom D, so by def'n of ACC, A ACC D.

4. **Chain condition:** For all nodes A, B, C, if A ACC C and B ACC C then A ACC B or B ACC A.

   Suppose A ACC C, and B ACC C.

   Then $\exists E$ such that E ImmDom B and E dom C and

[2]Note that the relation is transitive, so this representation is rather the intransitive reduction of the relation, just as syntax trees are the intransitive reduction of the transitive *dominates* relation. The real graph for the relation *asymmetrically c-commands* would have lines going directly from B to H and B to I.

$\exists F$ such that F ImmDom A and F dom C.

E and F both dominate C so one must dominate the other (by the chain condition on dom) (it doesn't matter which). Suppose E dom F.

Now, E ImmDom B and E dom F, and F ImmDom C. By transitivity of dom, E ImmDom B and E dom C, so B ACC C. ∎

Basically, ACC is a set of trees

4 Could you build a c-command set while you derive the sentence?

Looks like you can, provided:

1. You can give nodes unique labels as you go (addresses are defined from the top, which doesn't exist yet, so we need another way)
    - Possible if you keep a number running during the derivation, and you assign numbers to each node as you go.
2. you can tell what subtrees are movers
    - you know this from the form of the "state", or if you define the grammar to give the state as part of the output of each function, you don't need to compute it (I think??). Mover state is [Cat x; Neg y; ...]

How would this work?

- The derivation will not only perform the usual merge and move functions, but it will also give out a pair: (list of ACC trees, list of movers)
- The list of movers will be ordered triples: (last neg feature, root node label, list of ACC trees)
- So it kinda looks like this: (ACC trees, [(-f, root, ACC trees); (-g, root, AC trees);...])
- Leaves come with empty ACC list
- Merge:
    - **both non-movers**: 2 new ACC trees T1, T2: root node label of T1 dominates all trees in T2 ACC tree list. Root of T2 dominates all trees in T1 ACC tree list.
    - **mover merges with non-mover**: new ACC tree: non-mover dominates all non-mover trees (i.e. we get a trivial tree). Mover trees get added to second element of pair: mover list (last feature of mover, root node label of mover, list of ACCs already calculated)

- Move: Let O = the move node and let X = the node immediately dominated by the move node.
    1. When move cancels features, use that feature to find the right mover in the mover list
    2. new ACC tree: root of mover dominates ACC list trees from main derivation.
    3. second new ACC tree: X dominates ACC tree list from mover
- See attached example!