An introduction to mildly context sensitive grammar formalisms

— *The equivalence of TAGs and CCGs* —

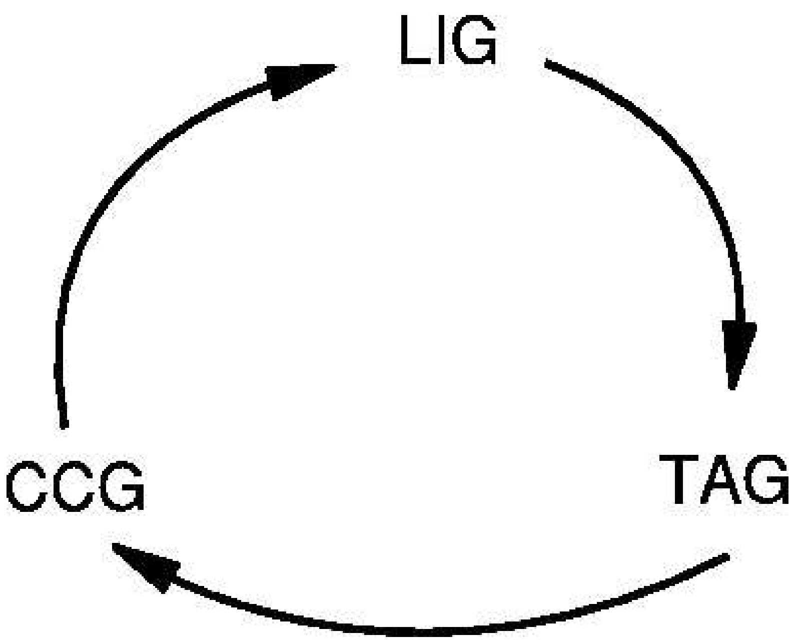
Gerhard Jäger & Jens Michaelis

University of Potsdam

{jaeger,michael}@ling.uni-potsdam.de

Overview

- CCGs and TAGs generate the same class of string languages
- can also be described by **Head Grammars** or **Linear Indexed Grammars**
- proper subset of the class of languages that is described by (set-local) **Multi-component TAGs** or **Linear Context-Free Rewriting Systems**
- proof: circular inclusion CCG $\rightarrow$ LIG $\rightarrow$ TAG $\rightarrow$ CCG



Overview

- Plan for this unit:
  - Indexed Grammars
  - Linear Indexed Grammars
  - CCG $\rightarrow$ LIG
  - LIG $\rightarrow$ TAG
  - TAG $\rightarrow$ CCG

Indexed Grammars

- generalization of CFGs
- strictly stronger than TAGs/CCGs
- introduced by Aho to handle variable binding in programming languages

Definition:
An **IG**, G, is denoted by

$$G = (V_N, V_T, V_S, S, P)$$

where

- V_N is a finite set of nonterminals
- V_T is a finite set of terminals
- V_S is a finite set of stack symbols
- V_N, V_T and V_S are mutually disjoint
- $S \in V_N$ is the start symbol, and

- P is a finite set of productions, having the following form.

$$A[\cdot\cdot x] \rightarrow \alpha_1 \ldots \alpha_n$$

where $x \in V_S^*$, and for each $1 \leq i \leq n$, $\alpha_i = A[\cdot\cdot y]$, $\alpha_i = A[z]$, or $\alpha_i = w$ where $A \in V_N$, $w \in V_T^*$, and $y, z \in V_S^*$.

Notational convention:

- $[\cdot\cdot l]$ ... arbitrary stack with l as top symbol

Comments:

- fixed number of symbols can be popped from LHS stack
- stacks of non-terminals on RHS:
  - fixed sized stack, or
  - unbounded stack from LHS, with a fixed number of symbols pushed on it
- notion of derivation ($\rightarrow_G^*$) is as in CFGs
- language $L(G)$ generated by the LIG G

$$L(G) = \{w \in V_T^* | S[] \rightarrow_G^* w\}$$

- example for a language that is generated by a LIG but not by a TAG/CCG:

$$a^n b^n c^n d^n e^n$$

- LIG that generates it:
  - $V_N = \{S, A, B, C, D, E\}$
  - $V_T = \{a, b, c, d, e\}$
  - $V_S = \{i\}$

- P :

$$
\begin{array}{rcl}
S[\cdot\cdot] & \longrightarrow & S[\cdot\cdot i] \\
S[\cdot\cdot\cdot] & \longrightarrow & A[\cdot\cdot\cdot]B[\cdot\cdot\cdot]C[\cdot\cdot\cdot]D[\cdot\cdot\cdot]E[\cdot\cdot\cdot] \\
A[\cdot\cdot ii] & \longrightarrow & aA[\cdot\cdot i] \\
A[i] & \longrightarrow & a \\
B[\cdot\cdot ii] & \longrightarrow & bB[\cdot\cdot i] \\
B[i] & \longrightarrow & b \\
C[\cdot\cdot ii] & \longrightarrow & cC[\cdot\cdot i] \\
C[i] & \longrightarrow & c \\
D[\cdot\cdot ii] & \longrightarrow & dD[\cdot\cdot i] \\
D[i] & \longrightarrow & d \\
E[\cdot\cdot ii] & \longrightarrow & eE[\cdot\cdot i] \\
E[i] & \longrightarrow & e
\end{array}
$$

Linear Indexed Grammar

- introduced by Gazdar (1985) for linguistic purposes
- proper restriction of IGs
- crucial innovation:
  - only one non-terminal on the RHS inherits the stack from the RHS
- dependencies between unbounded branches of a tree are not possible in LIGs

Example:

- previous example is not a LI language
- the following is a LI language though:

$$a^n b^n c^n d^n$$

- LIG that generates it:
  - $V_N = \{S, T\}$
  - $V_T = \{a, b, c, d\}$
  - $V_S = \{i\}$
  - P:

$$\begin{array}{rcl} S[\cdot\cdot] & \rightarrow & aS[\cdot\cdot i]d \\ S[\cdot\cdot] & \rightarrow & T[\cdot\cdot] \\ T[\cdot\cdot i] & \rightarrow & bT[\cdot\cdot]c \\ T[] & \rightarrow & \varepsilon \end{array}$$

From CCG to LIG

- first proved by Weir (1988)
- assumes particular format of CCG
  - no type lifting (can be done in the lexicon where needed)
  - only combinators: function application and (possibly mixed) function composition
  - applicability of combinators can be restricted to certain categories

Formal definition of CCG

A **CCG** G is denoted by (V_T, V_N, S, f, R), where

- V_T is a finite set of terminals (lexical items),
- V_N is a finite set of nonterminals (atomic categories)
- V_N and V_T are disjoint,
- S is a distinguished member of V_N,
- f is a function that maps elements of $V_T \cup \{\varepsilon\}$ to finite subsets of $C(V_N)$, the set of categories, where
  - $V_N \subseteq C(V_N)$, and if $c_1, c_2 \in C(V_N)$, then $(c_1/c_2) \in C(V_N)$ and $(c_1\backslash c_2) \in C(V_N)$
- R is a finite set of combinatory rules.

Formal definition of CCG

- four types of combinatory rules
- $x, y, z_1, ...$ variables over $C(V_N)$, $|_i$ is a variable over $\{/, \backslash\}$

1. forward application

$$(x/y)\ y \rightarrow x$$

2. backward application

$$y\ (x\backslash y) \rightarrow x$$

3. generalized forward composition: for some $n \geq 1$:

$$(x/y)\ (\ldots(y|_1 z_1)_2 \ldots |_n z_n) \rightarrow (\ldots(x_1|_1 z_1)|_2 \ldots |_n z_n)$$

4. generalized backward composition: for some $n \geq 1$:

$$(\ldots(y|_1 z_1)_2 \ldots |_n z_n)\ (x\backslash y) \rightarrow (\ldots(x_1|_1 z_1)|_2 \ldots |_n z_n)$$

Formal definition of CCG

- possible constraints on instantiations of variables:
  1. The initial nonterminal of the category to which x is instantiated can be restricted
  2. The entire category to which y is instantiated can be restricted.
- language $L(G)$ generated by CCG G:

$$L(G) = \{a_1 \ldots a_n | S \rightarrow^*_G c_1 \ldots c_n, c_i \in f(a_i), a_i \in V_t \cup \{\varepsilon\}, 1 \leq i \leq n\}$$

Note that empty categories are admitted as lexical entries.

Terminology:

- (x/y) in the forward rules and $(x\backslash y)$ in the backward rules is called the **primary category** of the rule.
- The other category is called the **secondary category** of the rule.

From CCG to LIG

- crucial observations:
  1. CCG categories can be seen as nonterminals + stack
     - for example:

$$\begin{array}{rcl} s & \leadsto & s[] \\ s/a & \leadsto & s[/a] \\ s/a\backslash b\backslash b/s & \leadsto & s[/a, \backslash b, \backslash b, /s] \\ s/(n\backslash s) & \leadsto & s[/(n\backslash s)] \end{array}$$

     - function application amounts to pushing item on stack
     - function composition is a combination of pushing and popping

- crucial observations:
  2. Each **component** of the RHS of a combinatory rule is also a component of one the LHS categories
     - set of components does not increase in syntactic composition
     - ultimately determined by lexicon
     - no upper limit for **number of components** in x in the combinatory rules
  3. For each combinatory rule, there are finitely many ground instances of the **secondary category**.
     - only x in the primary category has infinitely many instances
     - can be modeled by a LIG-stack

The construction

- Auxiliary notions:
  - τ maps a category to its **target**:

$$\begin{aligned} \tau(A) &= A \text{ if } A \in V_N \\ \tau(x/y) &= \tau(x) \\ \tau(x\backslash y) &= \tau(x) \end{aligned}$$

  - c maps a category to its **components**:

$$\begin{aligned} c(A) &= \{A\} \text{ if } A \in V_N \\ c(x/y) &= c(x) \cup \{y\} \\ c(x\backslash y) &= c(x) \cup \{y\} \end{aligned}$$

From CCG to LIG

- Auxiliary notions:
  - lexical components $\mathcal{C}$:

$$\mathcal{C} = \bigcup_{x \in rg(f)} c(x)$$

- translation tr from CCG categories to stacked LIG-categories:

$$\begin{aligned} tr(A) &= A[] \text{ iff } A \in V_N \\ tr(x/y) &= tr(x) + [/y] \\ tr(x\backslash y) &= tr(x) + [\backslash y] \end{aligned}$$

where $A[z] + \alpha = A[z, \alpha]$

From CCG to LIG

Let G be a CCG. We will construct an LIG G' which is weakly equivalent to G.

- $V'_T = V_T$
- $V'_N = V_N$
- $V_S = \{/x|x \in \mathcal{C}_G\} \cup \{\backslash x|x \in \mathcal{C}_G\}$
- for each ground instance of the secondary category in each combinatory rule $\alpha,\ \beta \to \gamma \in R$:

$$tr(\gamma) \to tr(\alpha),\ tr(\beta) \in R'$$

- for each $\langle \alpha, x \rangle \in f$:

$$tr(x) \to \alpha \in R'$$

Example: copy language

- CCG for copy language:
  - lexicon

$$
\begin{aligned}
f(a) &= \{S\backslash A/S, S\backslash A, A\} \\
f(b) &= \{S\backslash B/S, S\backslash B, B\}
\end{aligned}
$$

  - combinatory rules:

$$
\begin{aligned}
y\ (x\backslash y) &\rightarrow x \\
(x/S)\ (S\backslash z) &\rightarrow (x\backslash z) \\
(x/S)\ (S\backslash z_1/z_2) &\rightarrow (x\backslash z_1/z_2)
\end{aligned}
$$

Example: copy language

- corresponding LIG:

$$
\begin{array}{rcl}
S[\cdot\cdot] & \rightarrow & A\ S[\cdot\cdot\backslash A] \\
S[\cdot\cdot] & \rightarrow & B\ S[\cdot\cdot\backslash B] \\
S[\cdot\cdot\backslash A] & \rightarrow & S[\cdot\cdot/S]\ S[\backslash A] \\
S[\cdot\cdot\backslash B] & \rightarrow & S[\cdot\cdot/S]\ S[\backslash B] \\
S[\cdot\cdot\backslash A,/S] & \rightarrow & S[\cdot\cdot/S]\ S[\backslash A,/S] \\
S[\cdot\cdot\backslash B,/S] & \rightarrow & S[\cdot\cdot/S]\ S[\backslash B,/S] \\
S[\backslash A,/S] & \rightarrow & a \\
S[\backslash A] & \rightarrow & a \\
A & \rightarrow & a \\
S[\backslash B,/S] & \rightarrow & b \\
S[\backslash B] & \rightarrow & b \\
B & \rightarrow & b
\end{array}
$$

From LIG to TAG

- basic intuition:
  - LIG and TAG are analogous extensions of CFGs
  - CFGs: set of paths in a tree language is a regular language
  - LIGs/TAGs: set of paths is a context-free language
  - LIG : TAG = pushdown automaton : CFG in rewrite form

From LIG to TAG

- first step: normalize LIG
  - in normal form LIGs, every rule pushes or pops at most one item from the stack
  - straightforward to show that each LIG can be normalized (without changing the set of accepted strings)

From LIG to TAG

Let G be a LIG.

- in an LIG derivation, stacks are born
  - as empty stack at the root node of some derivation ($S[]$)
  - in the RHS of a rule (i.e. as a non-spinal daughter node)
- they die at the LHS of a lexical rule
- normalization: all stacks are born and die empty

The construction: Let G be a LIG. We want to construct an equivalent TAG G'.

- $V'_N = V_N \cup V_N \times ((V_S \times \{+,-\}) \cup \{\varepsilon\}) \times V_N$
- idea: adjunction nodes are labeled with elementary stack operations:
  - input nonterminal
  - transition type (no transition or pushing/popping one stack symbol)
  - output nonterminal
- start symbol remains the same

From LIG to TAG

- initial trees:

$$\begin{array}{c} A \\ | \\ [A\varepsilon B]/OA \\ | \\ B \end{array}$$

  for all nonterminals A, B

- $A \to x'$ for all rules $A[] \to x$ in R
  where x' is like x except that empty stacks are removed

From LIG to TAG

- auxilary trees: *for all nonterminals* A, B, C, D *and all stack symbols* a

$$\begin{array}{c} [A\varepsilon B]/NA \\ | \\ [A\varepsilon C]/OA \\ | \\ [C\varepsilon B]/OA \\ | \\ [A\varepsilon B]/NA \end{array} \qquad \begin{array}{c} [A\varepsilon B]/NA \\ | \\ [A + aC]/OA \\ | \\ [C\varepsilon D]/OA \\ | \\ [D - aB]/OA \\ | \\ [A\varepsilon B]/NA \end{array} \qquad \begin{array}{c} [A\varepsilon A]/NA \end{array}$$

From LIG to TAG

- further auxiliary trees:
  constructed from rules from G

$$A[\cdot\cdot] \to x\ B[\cdot\cdot]\ y \quad \rightsquigarrow \quad [A\varepsilon B]/NA \to x\ [A\varepsilon B]/NA\ y$$
$$A[\cdot\cdot] \to x\ B[\cdot\cdot a]\ y \quad \rightsquigarrow \quad [A + aB]/NA \to x\ [A + aB]/NA\ y$$
$$A[\cdot\cdot a] \to x\ B[\cdot\cdot]\ y \quad \rightsquigarrow \quad [A - aB]/NA \to x\ [A - aB]/NA\ y$$

From TAG to CCG

- first proved in Weir (1988)
- construction sketched here follows Vijay-Shanker and Weir (1994) (ftp://ftp.cogs.sussex.ac.uk/pub/users/davidw/mst94.pdf)
- basic idea: corresondence between TAG and CCG operations
    - substitution $\sim$ function application
    - adjunction $\sim$ function composition

From TAG to CCG

- footed tree (t, d):
  - d is address of a leaf of tree t
  - root of t and $d(t)$ have same label
  - spine: path from root to foot d
- normal form footed TAG trees (nfft):
  - at most binary branching
  - all internal nodes are either OA or NA
  - all OA-nodes are either on the spine or sister of nodes on the spine
- algorithm to transform nffts into CCG-categories:

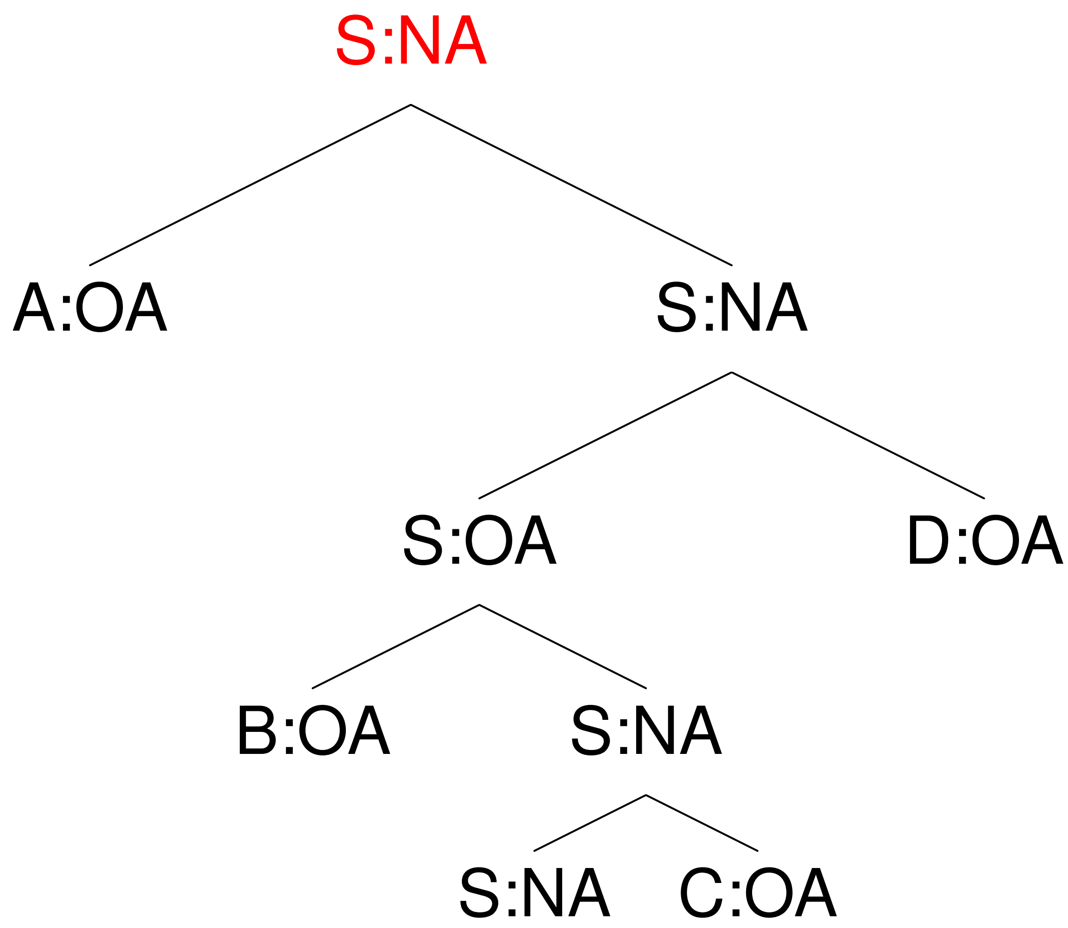
Algorithm nfft $(t, d) \rightsquigarrow$ **category**

- $pos =$ root of t
  - label(root(t)) = A
  - $c = A$ or $c = \hat{A}$
    ($x \mapsto \hat{x}$ is a bijection with range disjoint from $V_N \cup V_T$)

From TAG to CCG

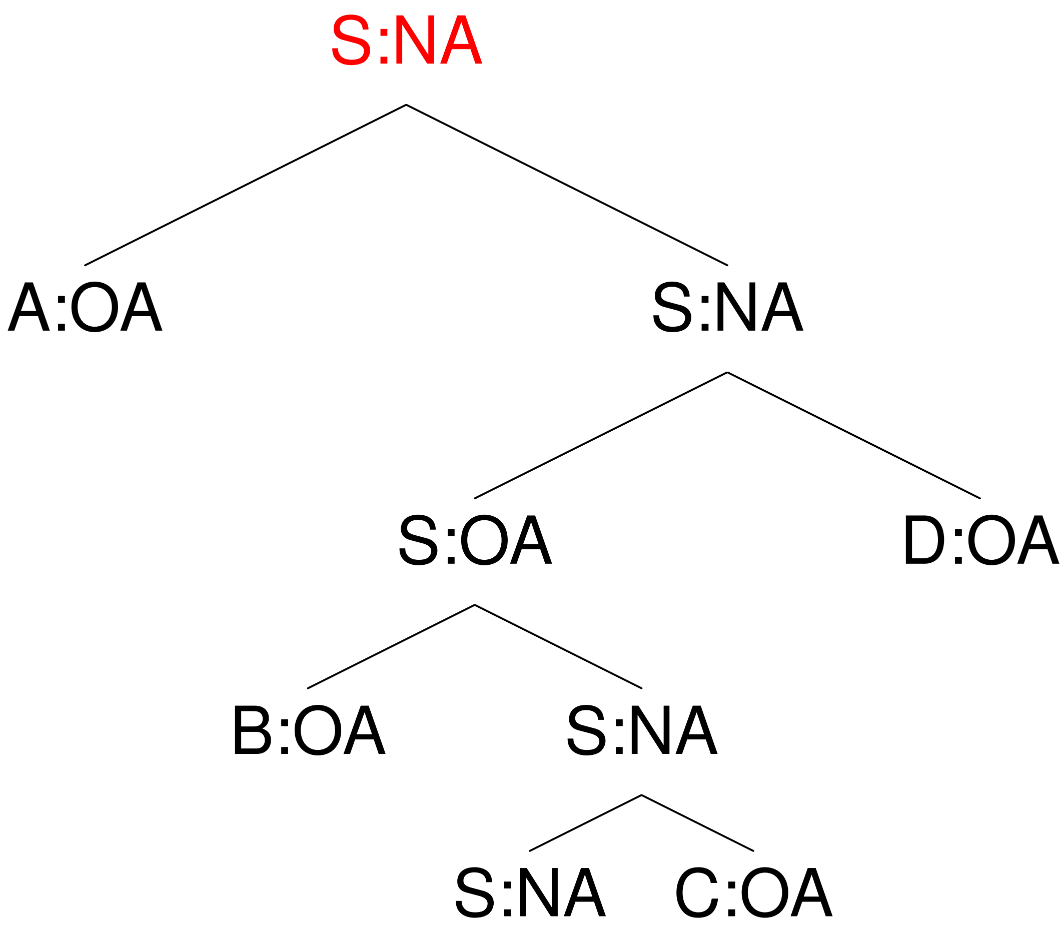
- until you reach d, do:
  - if the non-spine daughter of pos is a left daughter with label B/OA,
  $$c = c\backslash B$$
  - if the non-spine daughter of pos is a right daughter with label B/OA,
  $$c = c/B$$
  - if the spine daughter of pos has the label C/OA
  $$c = c/\hat{C}$$
  - $pos =$ spine daughter of pos

Example:



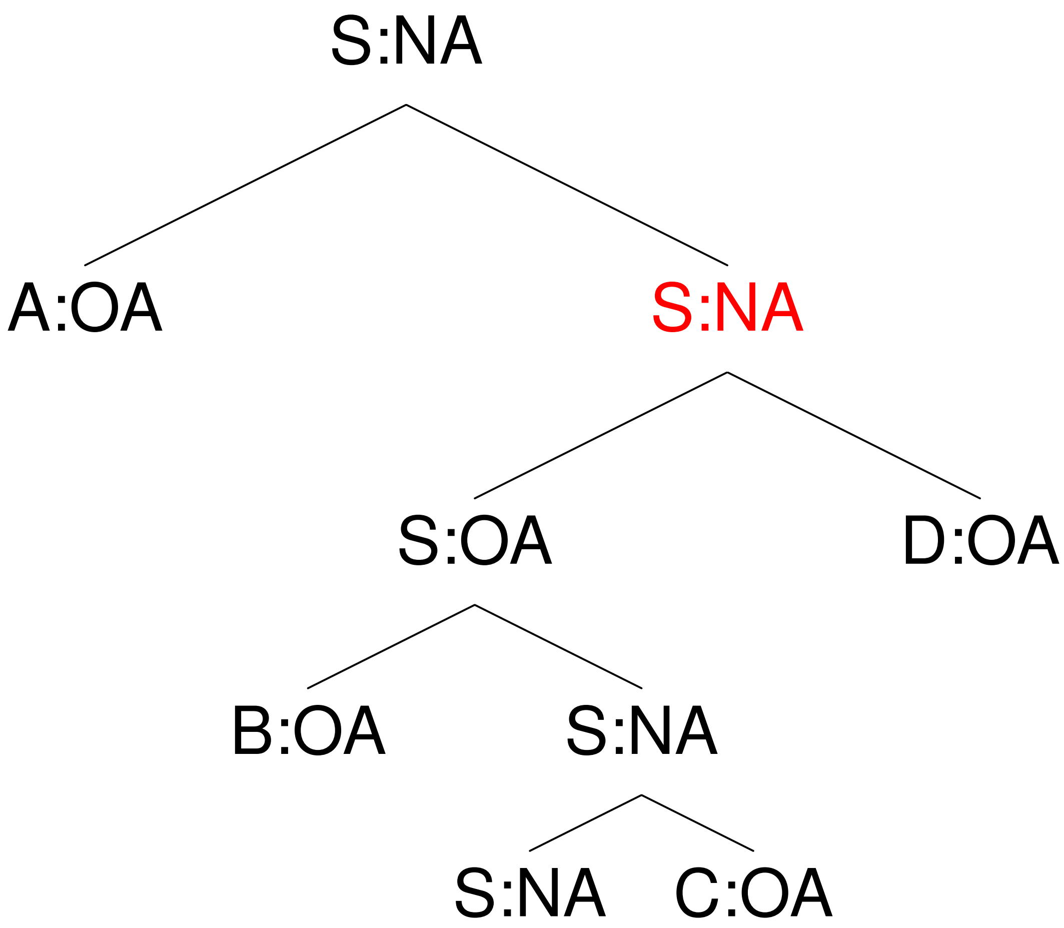
S $\hat{S}$

Example:



$S\backslash A$ $\hat{S}\backslash A$

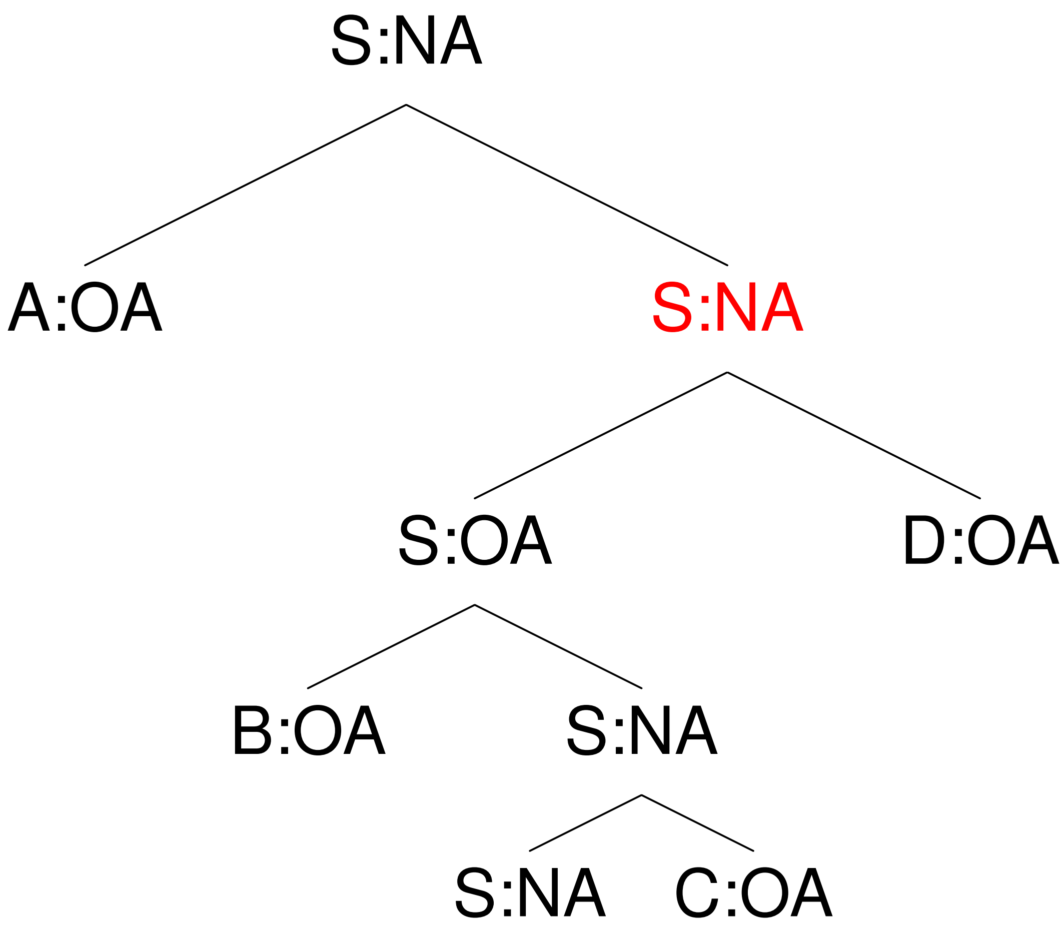
Example:



$S\backslash A$

$\hat{S}\backslash A$

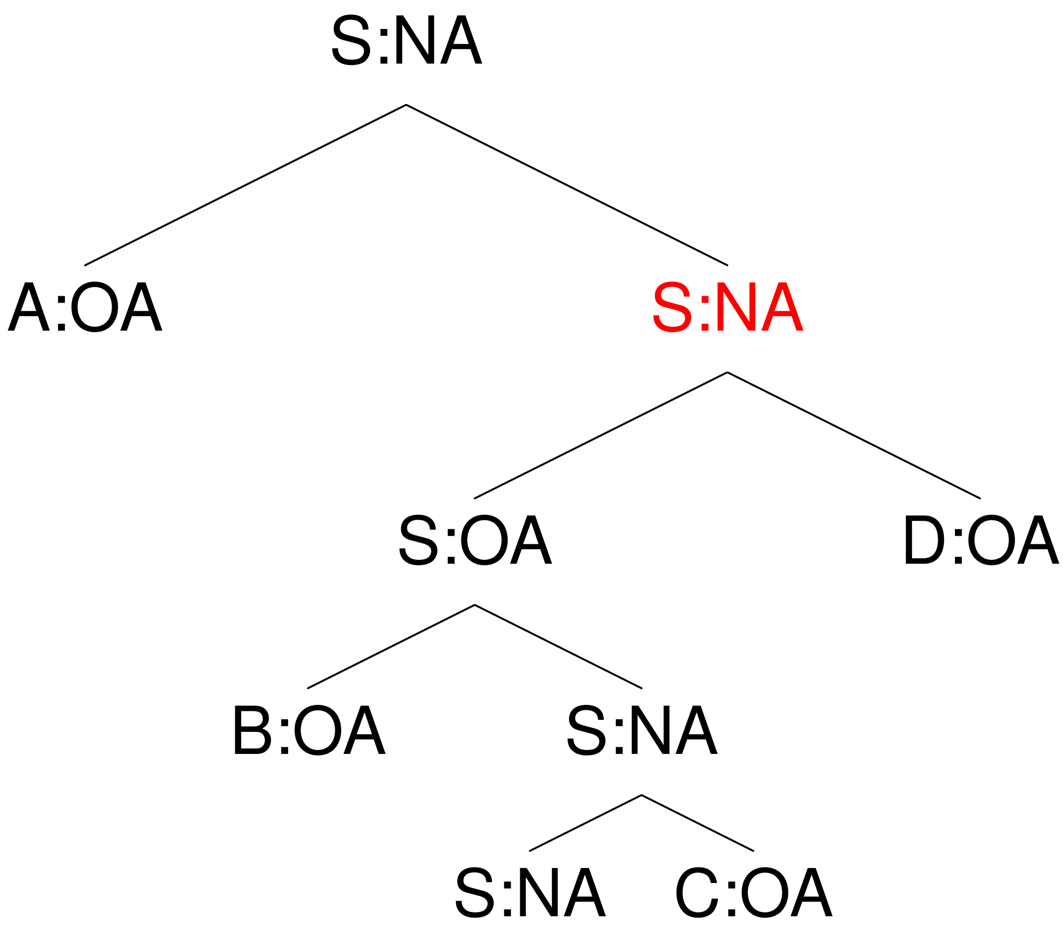
Example:



$S\backslash A/D$

$\hat{S}\backslash A/D$

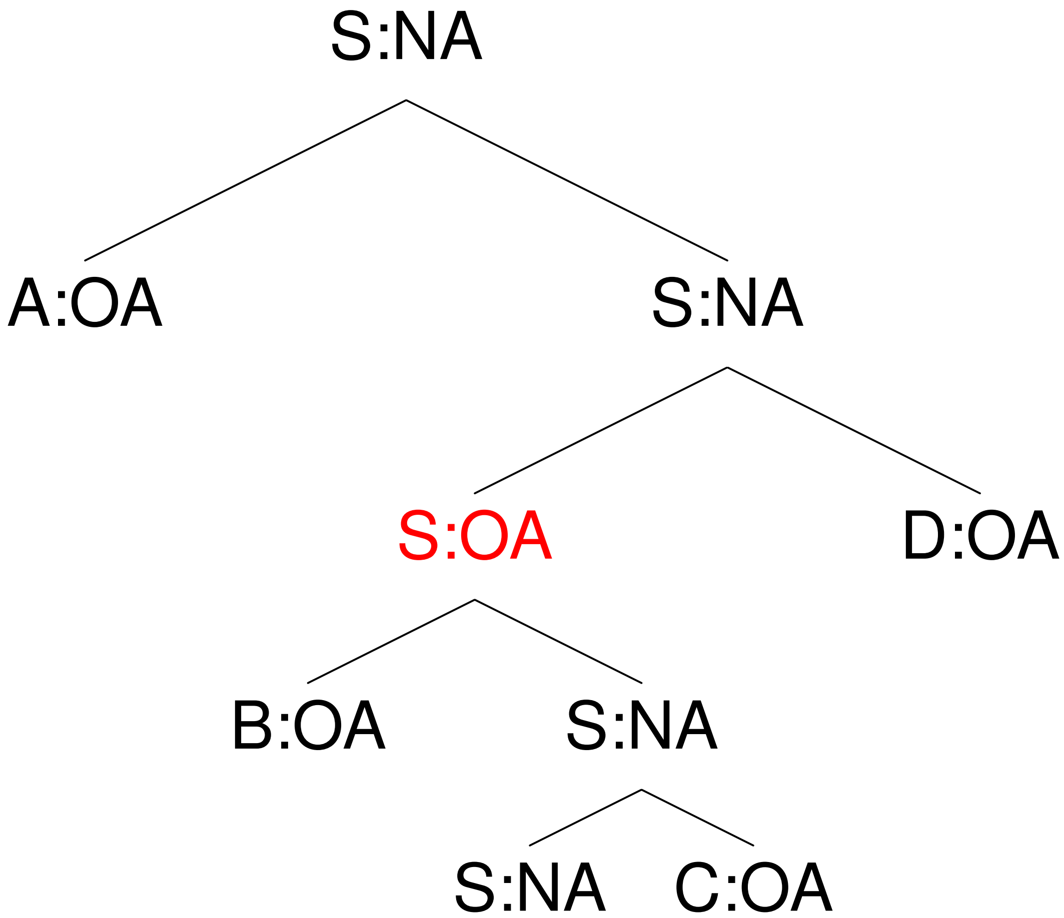
Example:



$S\backslash A/D/\hat{S}$

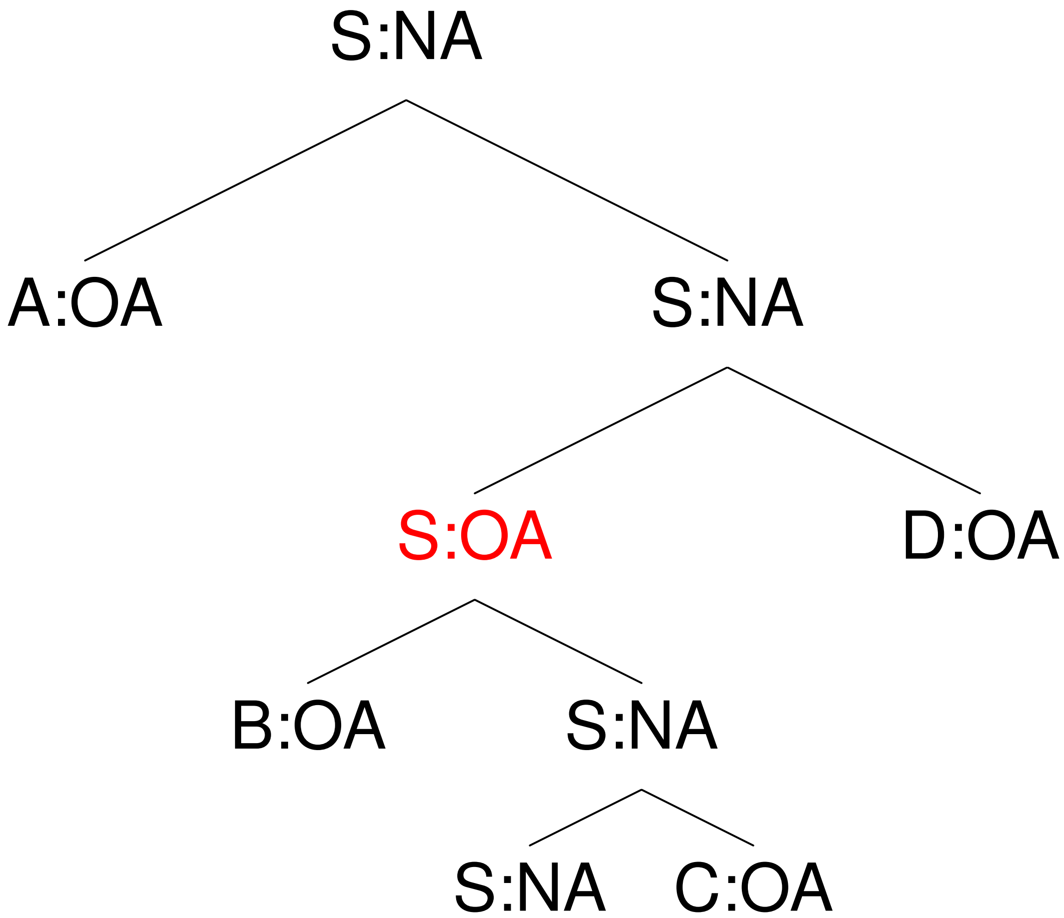
$\hat{S}\backslash A/D/\hat{S}$

Example:



$S\backslash A/D/\hat{S}$ $\hat{S}\backslash A/D/\hat{S}$

Example:

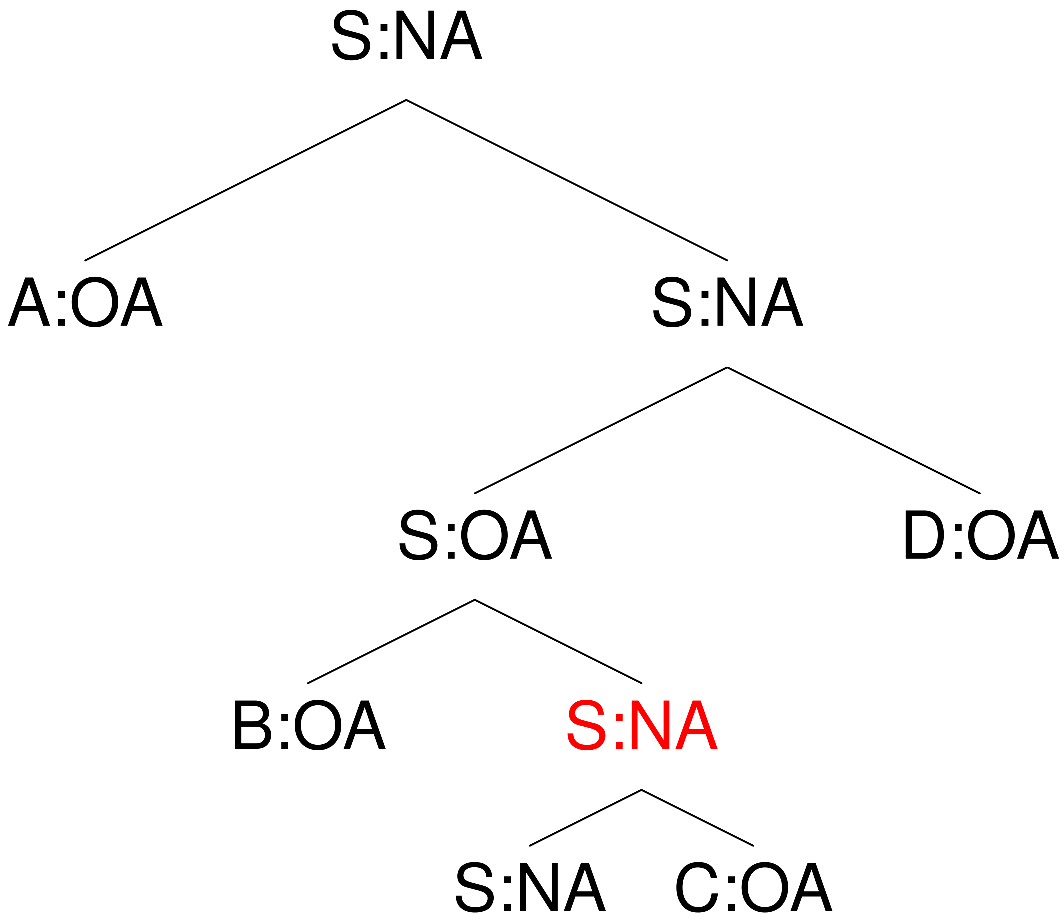


$S\backslash A/D/\hat{S}\backslash B$

$\hat{S}\backslash A/D/\hat{S}\backslash B$

From TAG to CCG

Example:

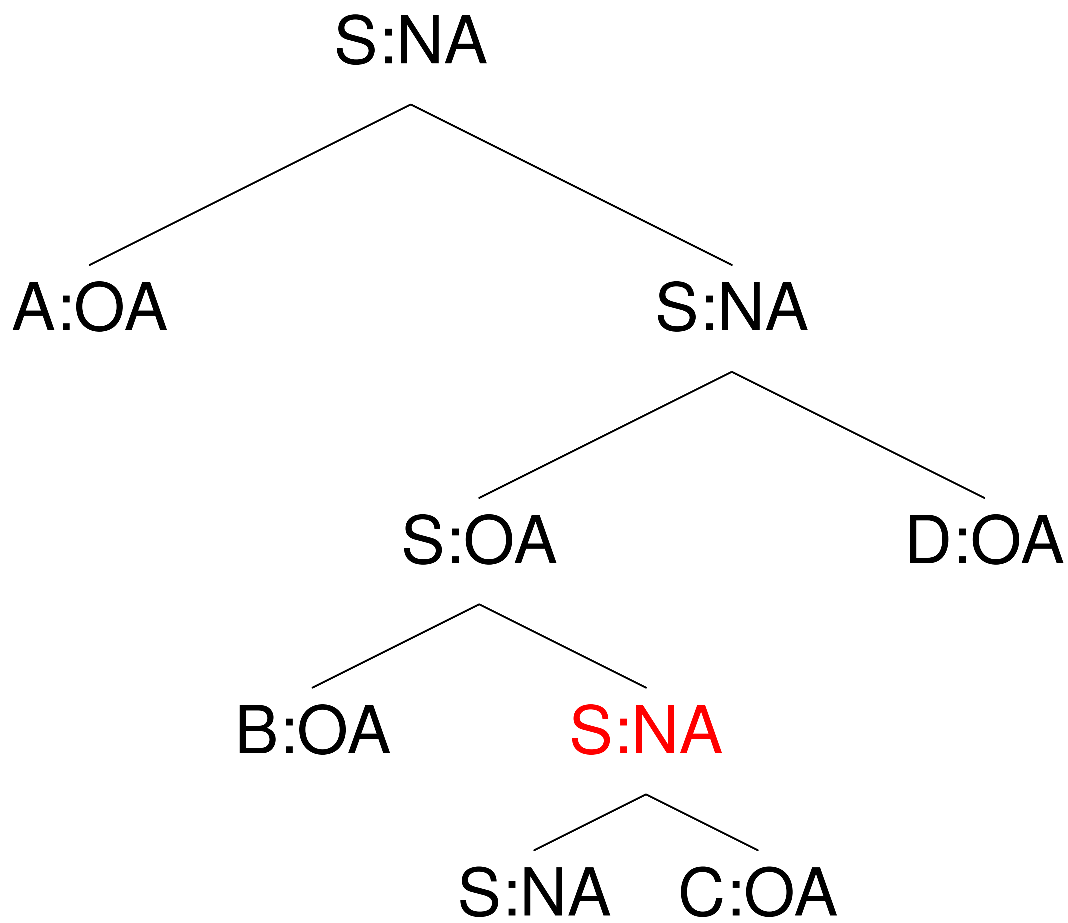


$S\backslash A/D/\hat{S}\backslash B$

$\hat{S}\backslash A/D/\hat{S}\backslash B$

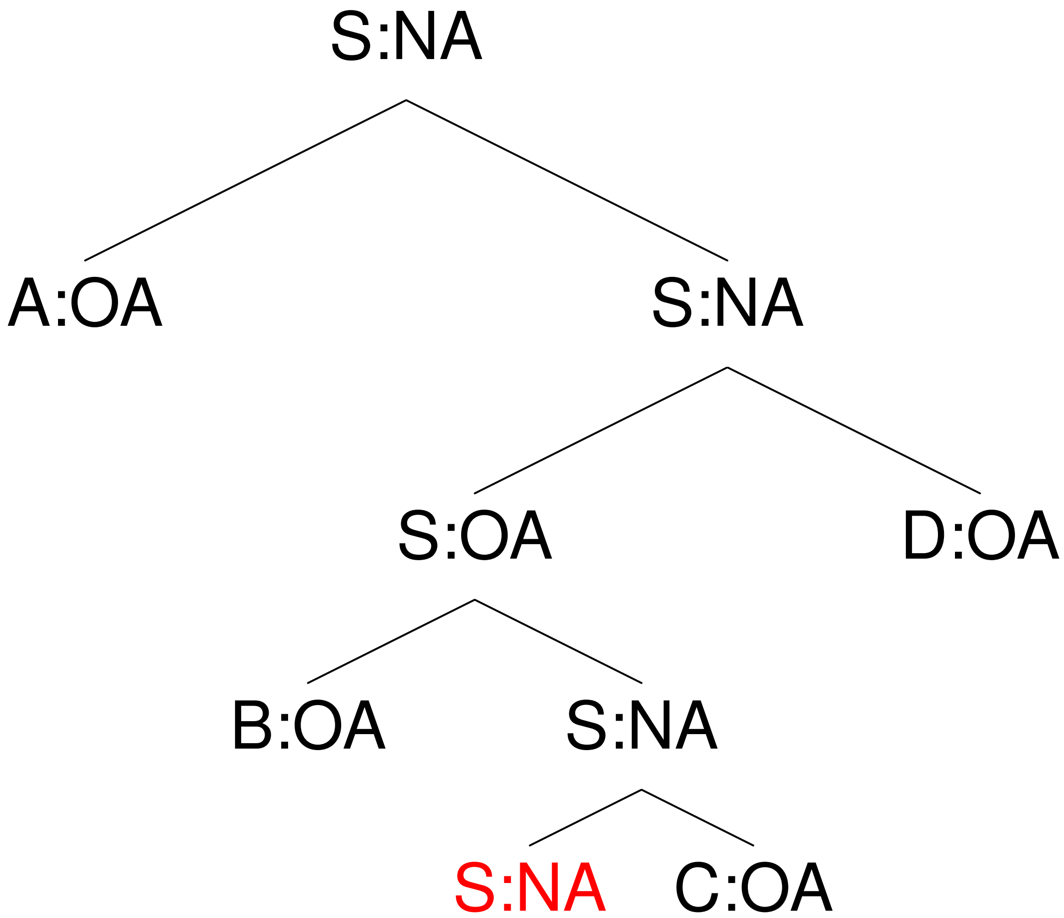
From TAG to CCG

Example:



$S\backslash A/D/\hat{S}\backslash B/C$ $\hat{S}\backslash A/D/\hat{S}\backslash B/C$

Example:



$S\backslash A/D/\hat{S}\backslash B/C$

$\hat{S}\backslash A/D/\hat{S}\backslash B/C$

From TAG to CCG

- 1-1 correspondence between nffts and corresponding categories
- in this fragment
  - function application corresponds to substitution, provided the substituted tree does only have NA-nodes
  - function composition corresponds to adjunction provided the yield of the adjoined tree is the empty string
- constraints can be enforced by using normal form TAGs

From TAG to CCG

Normal Form TAG

- initial trees are of the form:
  S is the start symbol, A is a non-terminal, w a terminal
  S does not occur as non-foot leaf

- auxiliary trees:
  - binary branching
  - all non-spine nodes are nonterminal leaves that are marked as OA

Observation:

All TAGs can be transformed into equivalent normal form TAGs.

Normal form derivation:

- adjoined tree is always an elementary tree
- adjunction/composition strictly bottom up:
  - the adjunction target does not dominate nonterminal leaves
  - all nonterminals dominated by the adjunction target are marked with NA
  - if the sister of the adjunction target is marked with OA, this sister is on the spine

Observation:

All trees that can be derived in a normal form TAG can be derived in a normal form derivation.

From TAG to CCG

The construction

Let G be a TAG in normal form. We construct an equivalent CCG G'.

- $V'_T = V_T$
- $V'_N = V_N \cup \{\hat{A} | A \in V_N\}$
- $S_G = S_{G'}$
- $A \in f(w)$ iff the following is an initial tree of G:

$$\begin{array}{c} \text{A} \\ | \\ \text{a} \end{array}$$

- $c \in f(\varepsilon)$ iff c is the result of transforming an auxiliary tree of G into a CCG category according the the algorithm above

From TAG to CCG

- rules of G':
  for each nonterminal A*, each* $i \leq n$*, where* n *is the maximal length of a spine of an auxiliary tree in* G*, and each* $|_j \in \{\backslash, /\}$

$$
\begin{array}{rcl}
(x/A)\ A & \rightarrow & x \\
A\ (x\backslash A) & \rightarrow & x \\
(x/\hat{A})(\ldots(\hat{A}|_1 z_1)|_2 \ldots |_i z_i) & \rightarrow & (\ldots(x|_1 z_1)|_2 \ldots |_i z_i)
\end{array}
$$