# 5. Combinatory Categorial Grammar (CCG) & Linear Indexed Grammar (LIG)

Page references are to (and screenshots of examples are from) Steedman 2000.

## 1 CCG basics

CCG is a "mostly lexicalized" formalism: each language specifies its own choice of categorized lexical items, plus its own restrictions on the set of allowable universal rules.

(1) *The functional application rules*

a. $X/Y \quad Y \Rightarrow X$ (>)

b. $Y \quad X\backslash Y \Rightarrow X$ (<)

(2) *Functional composition*

a. $X/Y \quad Y/Z \Rightarrow_{\mathbf{B}} X/Z$ ($>\mathbf{B}$)

b. $X/Y \quad Y\backslash Z \Rightarrow_{\mathbf{B}} X\backslash Z$ ($>\mathbf{B}_\times$)

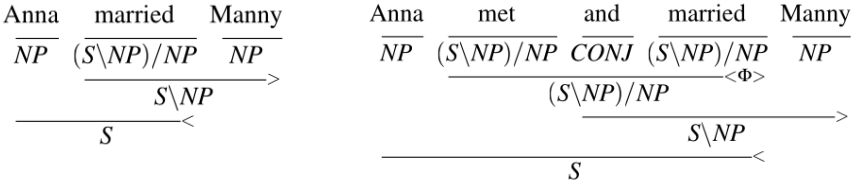
c. $Y\backslash Z \quad X\backslash Y \Rightarrow_{\mathbf{B}} X\backslash Z$ ($<\mathbf{B}$)

d. $Y/Z \quad X\backslash Y \Rightarrow_{\mathbf{B}} X/Z$ ($<\mathbf{B}_\times$)

(Plus certain generalizations of these composition rules for additional arguments — see e.g. (5) below.)

The basic function application rules just give us a system which is equivalent to a context-free grammar.

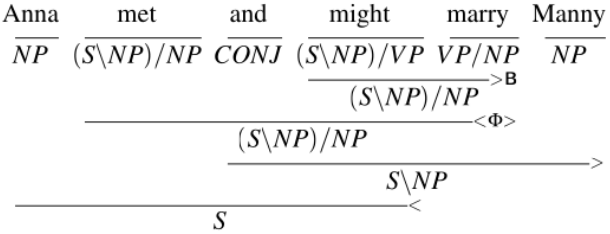
(3)



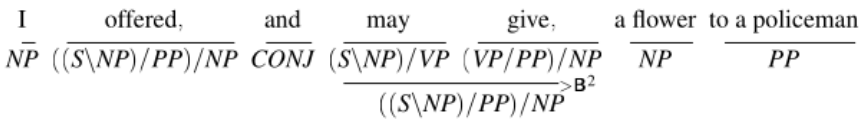

### 1.1 "Order-preserving" rules

The *non-crossed* composition rules are essentially order-preserving, but they allow for the creation of non-standard constituents. These play a role in explaining some kinds of coordination.

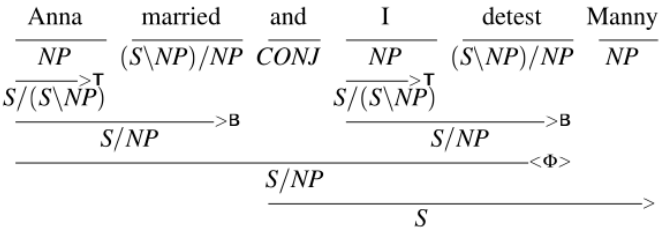
(4)



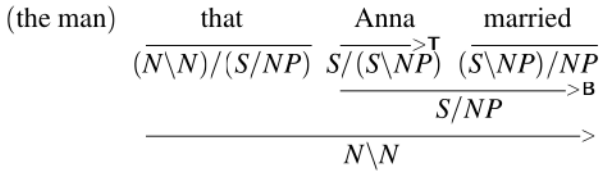
The generalized versions of composition allow more than one argument to be "delayed" — for example, an NP and a PP, instead of just an NP as above.

(5) 


In combination with *type-raising of arguments* (which Steedman would rather think of as a pre-syntactic, lexical operation), composition allows for "right node raising".

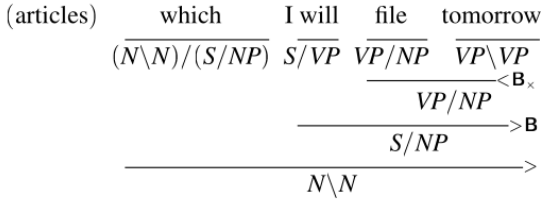
(6) 


The ability to make 'Anna married' a constituent here is also exactly what underlies the possibility of "extraction".

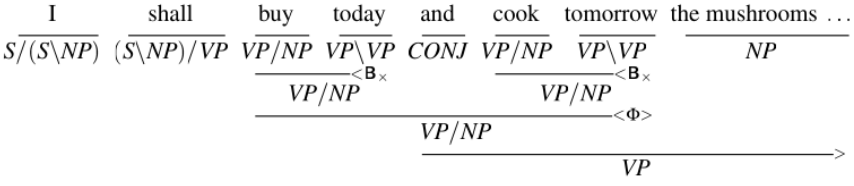
(7) 


## 1.2 Non-order-preserving (or "crossing") rules

A crossed composition rule is needed to derive extraction of any non-peripheral arguments: the crossed rule essentially makes the extracted argument peripheral, and then things work as before.

(8) 


The ability to make 'file tomorrow' a constituent here is also exactly what underlies heavy NP shift.

(9) 


These examples indicate that English includes a *backward crossed composition* rule (with some additional restrictions, see pp.63–64).

(10) *Backward crossed composition (preliminary version)* ($<\mathbf{B}_\times$)
$Y/Z \quad X\backslash Y \quad \Rightarrow_{\mathbf{B}} \quad X/Z$
where $X, Y = S\$$

But Steedman excludes the other logically-possible crossed composition rule — *forward crossed composition* — from English. This simultaneously rules out illicit subject extractions, and leftward "scrambling" (pp.59–60).

(11) $*\ X/Y \quad Y\backslash Z \quad \Rightarrow \quad X\backslash Z$

(12) a. a man who(m) [I think that]$_{S/S}$ [Dexter likes]$_{S/NP}$
b. * a man who(m) [I think that]$_{S/S}$ [likes Dexter]$_{S\backslash NP}$

(13) * I Dexter [think (that) likes Warren]$_{(S\backslash NP)\backslash NP}$

Notice the two important ways in which the two crossed composition rules that the theory allows (i.e. (10) and (11)) both "respect" the directionality of the categories involved (see p.54):

- "Consistency": In both cases, the direction in which the $Y$ is sought is the direction in which the $Z$-to-$Y$ thing is found.
- "Inheritance": In both cases, the direction in which the $Z$ is sought in the input categories is the direction in which it is sought in the output category.

So forward crossed composition is inherently about leftward "movement" of leftward arguments, and backward crossed composition is inherently about rightward "movement" of rightward arguments.

## 2 Dutch crossing dependencies in CCG

The standard German (context-free) nesting order is straightforward to derive with only function application (p.140).

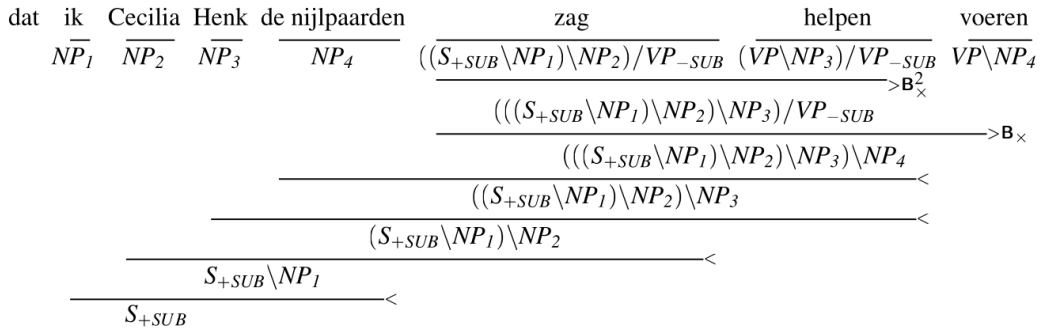
(14)
$$\dfrac{\dfrac{\dots\ \text{daß}}{S'_{+SUB}/S_{+SUB}}\ \dfrac{\text{ich}}{NP_1}\ \dfrac{\dfrac{\dfrac{\dfrac{\text{Cecilia}}{NP_2}\ \dfrac{\dfrac{\text{die Nilpferde}}{NP_3}\ \dfrac{\text{füttern}}{VP\backslash NP_3}}{VP}<}{(S_{+SUB}\backslash NP_1)\backslash NP_2}<}{S_{+SUB}\backslash NP_1}<}{S_{+SUB}}<\quad \dfrac{\text{sah}}{((S_{+SUB}\backslash NP_1)\backslash NP_2)\backslash VP}$$

The basic trick that allows the crossing dependencies pattern to fall out is the inclusion of *forward crossed composition* in Dutch (pp.141–142).

(15) *Dutch forward crossed composition I* ($>\mathbf{B}^n{}_\times$)
$X/Y \quad (Y\backslash Z)\$ \quad \Rightarrow_{\mathbf{B}^n} \quad (X\backslash Z)\$$
where $Y = VP_{-SUB}$

(16)
$$\dfrac{\dfrac{\dfrac{\dfrac{\text{dat}}{}\ \dfrac{\text{ik}}{NP_1}\ \dfrac{\dfrac{\dfrac{\dfrac{\text{Cecilia}}{NP_2}\ \dfrac{\text{de nijlpaarden}}{NP_3}}{((S_{+SUB}\backslash NP_1)\backslash NP_2)/VP_{-SUB}}\ \dfrac{\text{zag}}{}}{((S_{+SUB}\backslash NP_1)\backslash NP_2)\backslash NP_3}>\mathbf{B}_\times}{(S_{+SUB}\backslash NP_1)\backslash NP_2}<}{S_{+SUB}\backslash NP_1}<}{S_{+SUB}}<\quad \dfrac{\text{voeren}}{VP\backslash NP_3}$$

(17)




The function composition rules combine the verbs into a single constituent which has *accumulated* all of the verbs' argument-taking slashes.

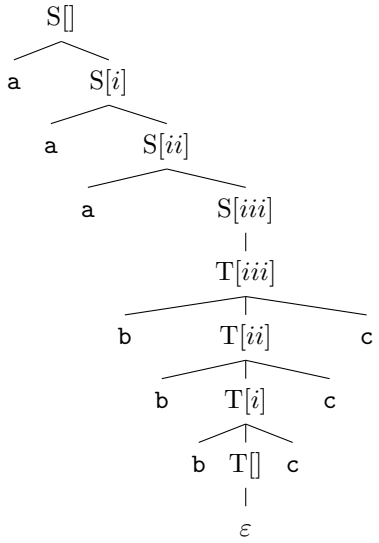
# 3 Linear Indexed Grammar

A Linear Indexed Grammar (LIG) enriches the now-familiar machinery of a CFG(/SLTG/FSTA) with a stack-based memory.[1] Each rule can pass the stack associated with its left-hand side nonterminal on to *at most one*[2] of its daughter nonterminals on the right-hand side, with the option of pushing and/or popping some stack symbols as it does so.

Here's an example grammar. The top of the stack is on the right. The alphabet is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, the nonterminals are $\{\mathrm{S}, \mathrm{T}\}$, and the stack alphabet (or the set of "indices") is $\{i\}$.

(18)
$\mathrm{S}[\cdots] \to \mathtt{a}\ \mathrm{S}[\cdots i]$
$\mathrm{S}[\cdots] \to \mathrm{T}[\cdots]$
$\mathrm{T}[\cdots i] \to \mathtt{b}\ \mathrm{T}[\cdots]\ \mathtt{c}$
$\mathrm{T}[] \to \varepsilon$

(19)




What class of languages could we generate if we restricted ourselves to left-branching LIG rules?

**Useful practice:**
- Write a Linear Indexed Grammar for the (non-context-free) `flip/flop/tick/tock` language with crossing dependencies.
- Write a Linear Indexed Grammar for the (non-context-free) language $\{ww \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$.

[1] Gazdar 1988 is a nice introduction to LIGs.
[2] Without this "at most one" restriction, we have the more general class of Indexed Grammars.

# 4 Equivalence of CCG and LIG

The basic idea of the correspondence between CCG and LIG[3] is that the arguments required by a complex category are represented as symbols on the stack.

(20) $(VP/NP_2)/NP_1 \quad \equiv \quad VP[NP_2, NP_1]$

(21) $(((S/NP_4)/NP_3)/NP_2)/NP_1 \quad \equiv \quad S[NP_4, NP_3, NP_2, NP_1]$

Then two simple application and composition rules look like this (p.210):

(22)
$$\begin{array}{llllll} (X\cdots)/Y & Y & \Rightarrow & (X\cdots) & \equiv & X[\cdots] \rightarrow X[\cdots Y]\ \ Y[] \\ (X\cdots)/Y & Y/Z & \Rightarrow & (X\cdots)/Z & \equiv & X[\cdots Z] \rightarrow X[\cdots Y]\ \ Y[Z] \end{array}$$

The LIG's unbounded stack is letting us accommodate the fact that there are unboundedly many possible choices for $(X\cdots)$, i.e. the range of the "principal function".

The equivalence of LIG and CCG relies on the fact that that position — the range of the principal function — is the *only* place where unboundedness can accumulate.

For this to be the case, we need type-raising to be finitely constrained (i.e. effectively, it's just lexical ambiguity rather than a syntactic operation), and we need to be careful about generalizing the composition rules.

## 4.1 Generalized composition

Recall that we need generalized versions of the composition rules that allow for multiple arguments to be "delayed". Here's one of $B_2$ rules, for example.

(23) $(X\cdots)/Y \quad (Y/Z_2)/Z_1 \quad \Rightarrow \quad ((X\cdots)/Z_2)/Z_1 \quad \equiv \quad X[\cdots Z_2 Z_1] \rightarrow X[\cdots Y]\ \ Y[Z_2, Z_1]$

- This is compatible with LIG's one-daughter requirement, because only the $(X\cdots)$ component is unbounded.
- A completely general $B^n$ rule with an unbounded number of delayed arguments $Z_1, Z_2, Z_3, \cdots$, however, *cannot* be recast as an LIG rule: such a rule would pass an unbounded amount of information down to *two* daughter nonterminals. Steedman assumes there is some maximum $n$ for each language (e.g. $n \leq 3$ for English, pp.42–43).
- This limit does not put a bound on the crossing-dependencies construction: the derivation in (17) will generalize up to arbitrarily many verbs using only $>B^2_\times$. The $>B^2_\times$ rule in (17) allows for $(\cdots\backslash NP_3)/VP_{-SUB}$ to be "passed along", and the additional step in a four-verb derivation would likewise pass along $(\cdots\backslash NP_4)/VP_{-SUB}$, and so on. So just the two elements $Z_1$ and $Z_2$ in (23) suffice.
- The kind of derivation in Steedman's (16) on p.141, however, where the verb cluster effectively has a right-branching structure, will *not* generalize up unboundedly: that three-verb derivation requires $>B^2_\times$ (to allow $(\cdots\backslash NP_3)\backslash NP_4$ to be passed along) and a corresponding example with four verbs would require $>B^3_\times$ (to allow $((\cdots\backslash NP_3)\backslash NP_4)\backslash NP_5$ to be passed along).

## 4.2 An abstract illustration

It should be easy to see how the LIG in (18) could be extended to a grammar for $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n \mid n \geq 0\}$, which is a kind of "prototypical" LIG language.

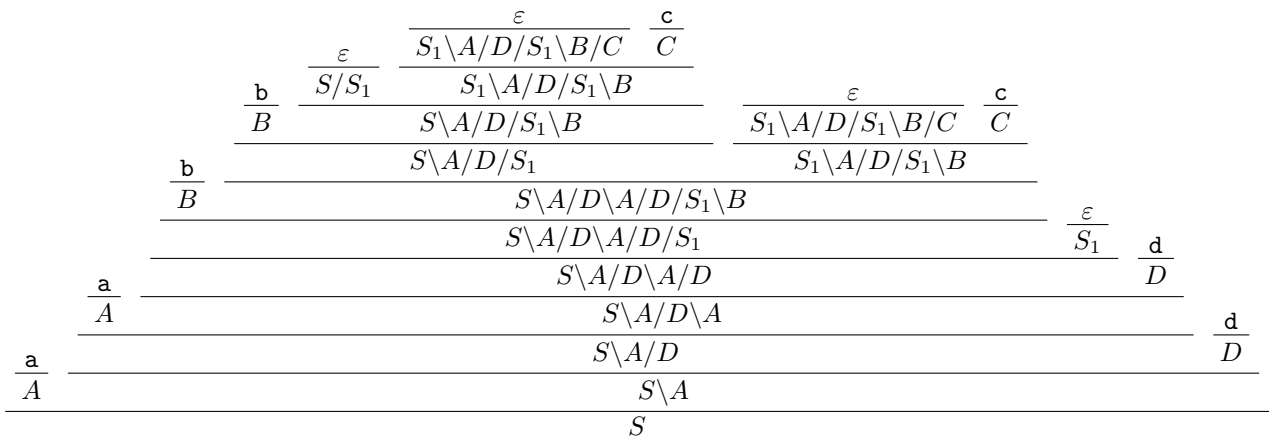
[3]For more detail, see Steedman 2000, ch.8, or Joshi et al. 1990, sec.5.

From Joshi et al. 1990, pp.41–42, here's a corresponding CCG that brings out the relationship between the two formalisms.

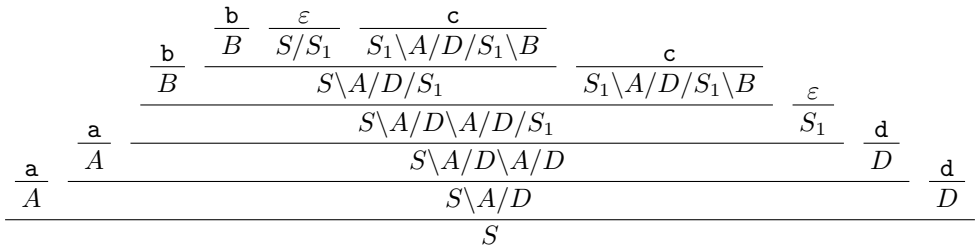
(24)

| | | | | |
|---|---|---|---|---|
| $\texttt{a} :: A$ | $(S_1 \cdots)/C \quad C$ | $\Rightarrow$ | $(S_1 \cdots)$ | |
| $\texttt{b} :: B$ | $(S \cdots)/S_1 \quad S_1$ | $\Rightarrow$ | $(S \cdots)$ | |
| $\texttt{c} :: C$ | $(S \cdots)/D \quad D$ | $\Rightarrow$ | $(S \cdots)$ | |
| $\texttt{d} :: D$ | $A \quad (S \cdots)\backslash A$ | $\Rightarrow$ | $(S \cdots)$ | |
| $\varepsilon :: S/S_1$ | $B \quad (S \cdots)\backslash B$ | $\Rightarrow$ | $(S \cdots)$ | |
| $\varepsilon :: S_1$ | $(S \cdots)/S_1 \quad S_1\backslash z_1/z_2/z_3\backslash z_4$ | $\Rightarrow$ | $(S \cdots)\backslash z_1/z_2/z_3\backslash z_4$ | |
| $\varepsilon :: S_1\backslash A/D/S_1\backslash B/C$ | | | | |

Here's the derivation of `aabbccdd`:




Zooming out a bit, here's the important structure in a (possibly?) clearer form:




# References

Gazdar, G. (1988). Applicability of indexed grammars to natural languages. In Reyle, U. and Rohrer, C., editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company, Dordrecht.

Joshi, A. K., Shanker, K. V., and Weir, D. (1990). The convergence of mildly context-sensitive grammar formalisms. University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-01, `https://repository.upenn.edu/cis_reports/539/`.

Steedman, M. (2000). *The Syntactic Process*. MIT Press, Cambridge, MA.