

LING419 Homework 1

Categorial Grammar Basics and Lambda Notation

Categorial Grammar (11 points)

When you give derivations, label each step with the rule you're using to make that step. You can use whatever notation you want for this, as long as it's understandable, but the simplest might be to adopt Steedman's: $<$ and $>$ for function application, $<B$ and $>B$ for function composition, and T for type-raising.

- (1) Give derivations for the following sentences. (Note: S' and S are two different categories.)
 - a. John $[\text{should}]_{(S \backslash NP)/(S \backslash NP)}$ $[\text{live}]_{(S \backslash NP)/PP}$ in Edinburgh
 - b. Mary $[\text{introduced}]_{((S \backslash NP)/PP)/NP}$ John to Susan
 - c. Mary $[\text{believes}]_{(S \backslash NP)/S'}$ $[\text{that}]_{S'/S}$ John left
 - d. John met, and I $[\text{think}]_{(S \backslash NP)/S'}$ that Mary will like, Bill
- (2) Although we haven't dealt with it in class, simple nouns like 'cat' and 'dog' have category N . Give derivations showing that the following "chunks"/phrases also have category N .
 - a. $[\text{big}]_{N/N}$ $[\text{red}]_{N/N}$ ball
 - b. man $[\text{who}]_{(N \backslash N)/(S \backslash NP)}$ John met
- (3) In (1c) we used the category $(S \backslash NP)/S'$ for the word 'believe', and used the category S'/S for the word 'that'. Suppose someone suggests that we could have avoided introducing the new category S' , and instead just used $(S \backslash NP)/S$ for 'believe' and S/S for 'that'.
 - a. Show that this suggestion will work for the sentence in (1c).
 - b. Show that this suggestion sometimes causes problems, by demonstrating that *if* we adopted it there would be certain non-sentences that our theory would incorrectly predict to be acceptable sentences. (You'll have to think up one of these "non-sentences" yourself.)
 - c. Now consider the sentence 'Mary believes John left'. Which theory deals with this sentence better — the original version from (1c) with the S' category, or the suggested alternative without it?

Lambda notation (9 points)

In this section, what matters is the "lambda-manipulating" steps — not any of the extra steps where it might be possible to further simplify the thing you end up with when all the "lambda work" is done. So for example, if you're asked to reduce $(\lambda x.x + 3)(5)$, all that matters is getting from there to $5 + 3$. It doesn't matter at all whether you leave it as $5 + 3$ or turn that into 8.

- (4) Reduce the following lambda expressions.

- a. $((\lambda x \lambda y. x^2 + 3xy + 4)(1))(3)$
- b. $(\lambda x \lambda y. x^2 + 3xy + 4)(2)$
- c. $(\lambda y. met'(y)(john'))(mary')$
- d. $(\lambda f. f(3))(\lambda w. w + 2)$
- e. $(\lambda f. f(3))((\lambda x \lambda y. x^2 + 3xy + 4)(1))$
- f. $(\lambda f. f(mary'))(\lambda x. walks'(x))$

(5) Consider the following function: $\lambda f \lambda x \lambda y. 2 \times f(x)(y)$

This takes a function and two numbers, applies the function to the two numbers and then doubles the result. (Take a minute to convince yourself that this is what it does.) Let's give it a sensible name:

$$ApplyThenDouble' = \lambda f \lambda x \lambda y. 2 \times f(x)(y)$$

Now reduce the following lambda expressions. Show all the intermediate steps, even though you might be able to see what the result will be relatively quickly.

- a. $(ApplyThenDouble'(\lambda a \lambda b. 3a + 2b))(5)(3)$
- b. $(ApplyThenDouble'(\lambda p \lambda q. p - q))$
- c. $(ApplyThenDouble'(\lambda w \lambda x. w + x))(2)(3)$ (Careful!)