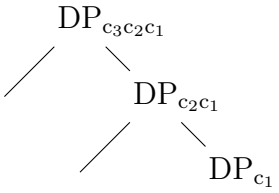
Computational Syntax Notes

09 September 2019

In 2007, we got CCG Bank, which has been used for machine learning and, paired with a neural-network supertagger, A* parsing. More recently, MG Bank came out so that people can do the same things with MGs instead of CCGs.

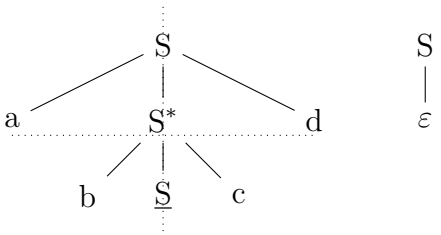
Regular languages can capture any finite language, but their weakness is that they can't capture dependencies that occur in patterns like $a^n b^n$. This pattern in particular is a *Counter Language*, meaning it can be captured by what is effectively a regular language paired with a single integer counter. There exist Context-Free Languages that are not Counter Languages, though. One example is the palindrome language, ww^R. A Recursive Transition Network with unbounded recursion is equivalent to a Context-Free Grammar (CFG), so such networks produce Context-Free Languages.

For some patterns, even a CFG is not sufficient. These include the copy language, ww, that we use to analyze Swiss German crossing dependencies ($NP_1NP_2NP_3VP_1VP_2VP_3$), or Old Georgian case stacking:



The latter does not even have the "constant-growth property", as for the n$^{\text{th}}$ DP, the string becomes n markers longer.

The copy language is not Context-Free, but it is Tree-Adjoining. We have analyzed this case before. Another Tree-Adjoining Languages is $a^n b^n c^n d^n$:



Here you get to split the tree into quadrants. This doesn't work for $a^n b^n c^n d^n e^n$, so the latter is not Tree-Adjoining — there's just nowhere to splice in the e leaves.

At one point there was a proof that CFGs can produce all and only the Tree-Adjoining Languages, but that relied on assumptions that really don't hold:

- The existence of empty lexical items (i.e. unpronounced material), and
- A lack of type-raising and multi-modality.

So we don't really know where CFGs fall.

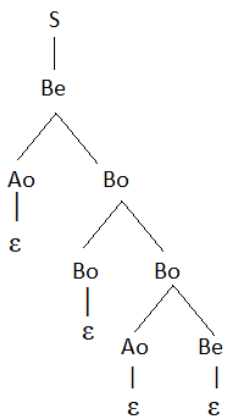
Beyond Tree-Adjoining Languages, there are Multiple Context-Free Languages (MGs without copying) and Parallel Multiple Context Free Languages (MGs with copying). One way to think about the distinctions between all these classes is slicing points:

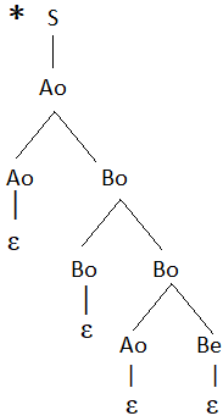
- Regular languages can append at one end or the other of a string.
- Context-Free can slice in one point and insert things there.
- TAL (TAG) can slice at up to three points and insert in each.
- MCFL (MG) can have finitely-many slices.
- PMFCL is weird.

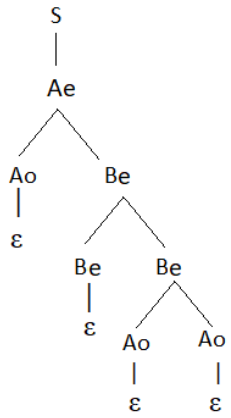
Computational Syntax class notes Monday (09/09/2019)

The CFG that only generate Ae non-terminals:
(Note: “Ae” refers to even numbers of A; “Ao” refers to odd numbers of A)

S —> Ae|Be
Ae —> AoAe|AeAo|BoAe|BeAo|AoBe|AeBo|BoBe

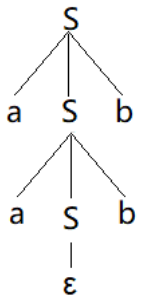






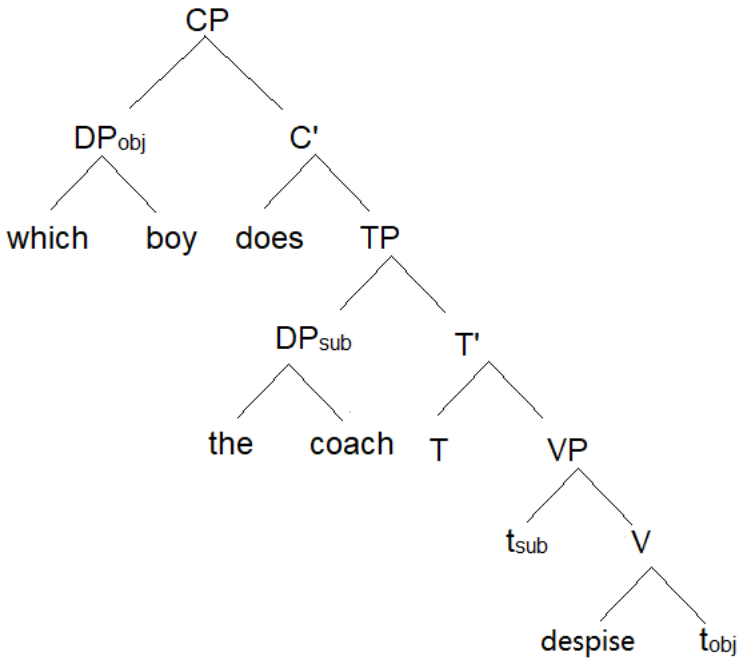
SL-2 (Strictly two local):
Example: n -> m| _{p,b} {np, nb} *a**nb**ar

CFG:
Example: S -> aSb, S -> ε

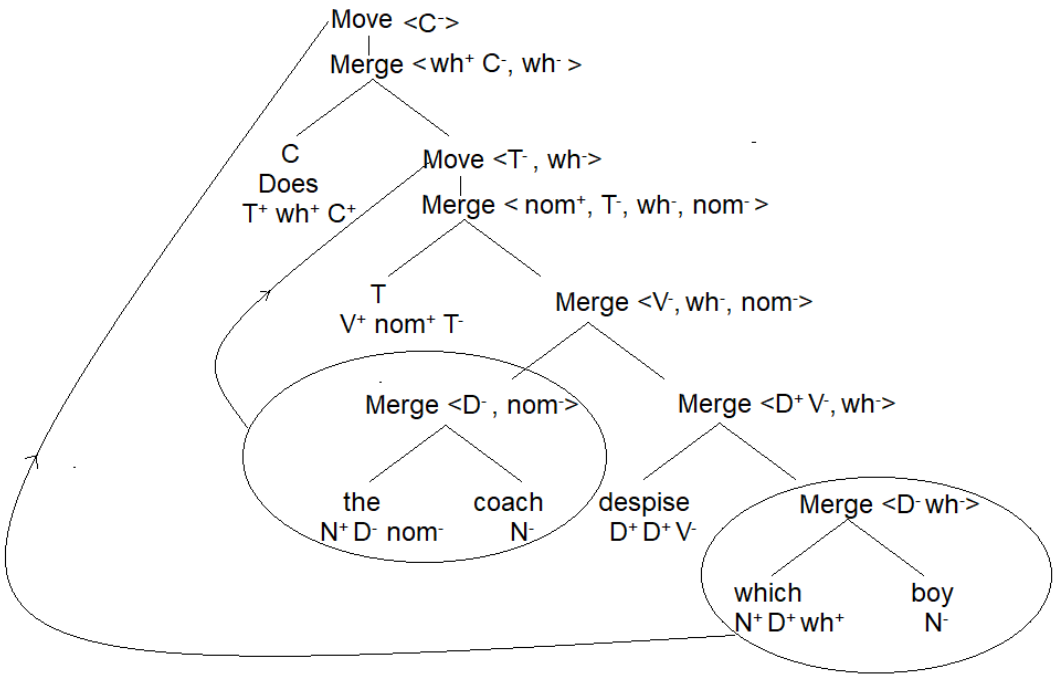


SL-2 <—cylindrification— REG
v v
| | string-yield
CFG ————————> CFL

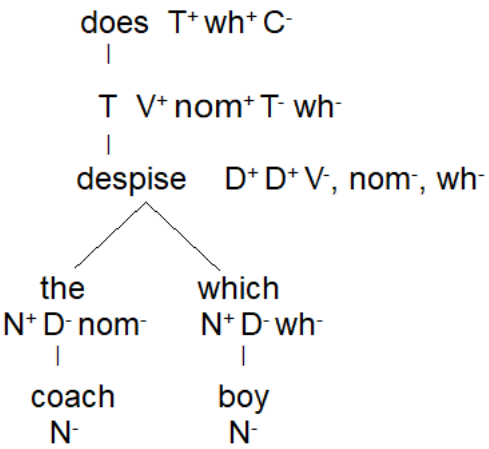
Phrase Structure tree:



Derivation Tree:

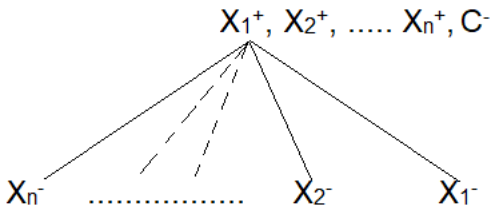


Dependency Tree:

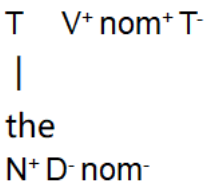


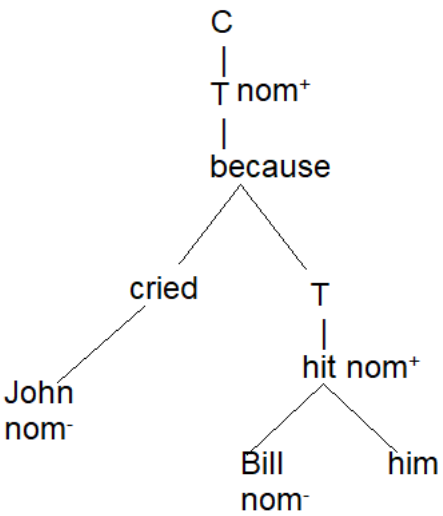
$(X^{+}\ \{f^{+}, y^{+}\}^{*}\ )\ C^{-}\ f^{-}\ g^{-}\ h^{-}$

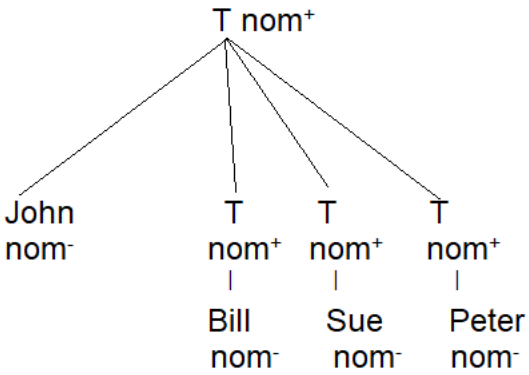
Merge:

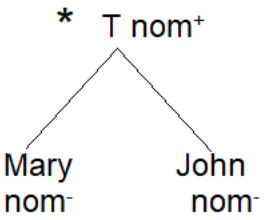


TSL (Tier-strictly Local):
nom tier:









Two constraints for movement (nom):

1. Every nom$^-$ tier has one nom$^+$ mother;
2. Every nom$^+$ mother has one nom$^-$ daughther.