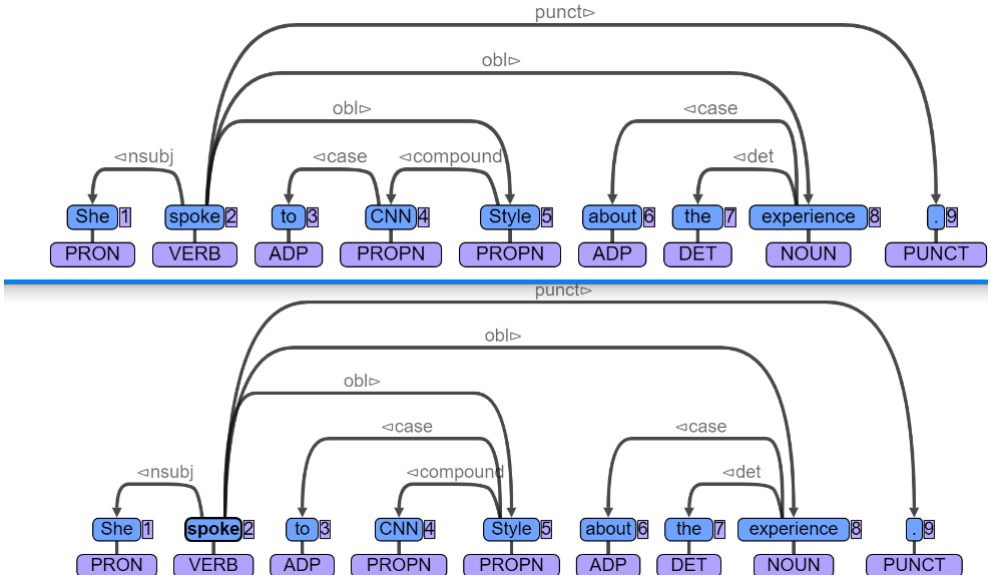
Assignment 3: Syntax

Oreen Yousuf

December 7, 2020

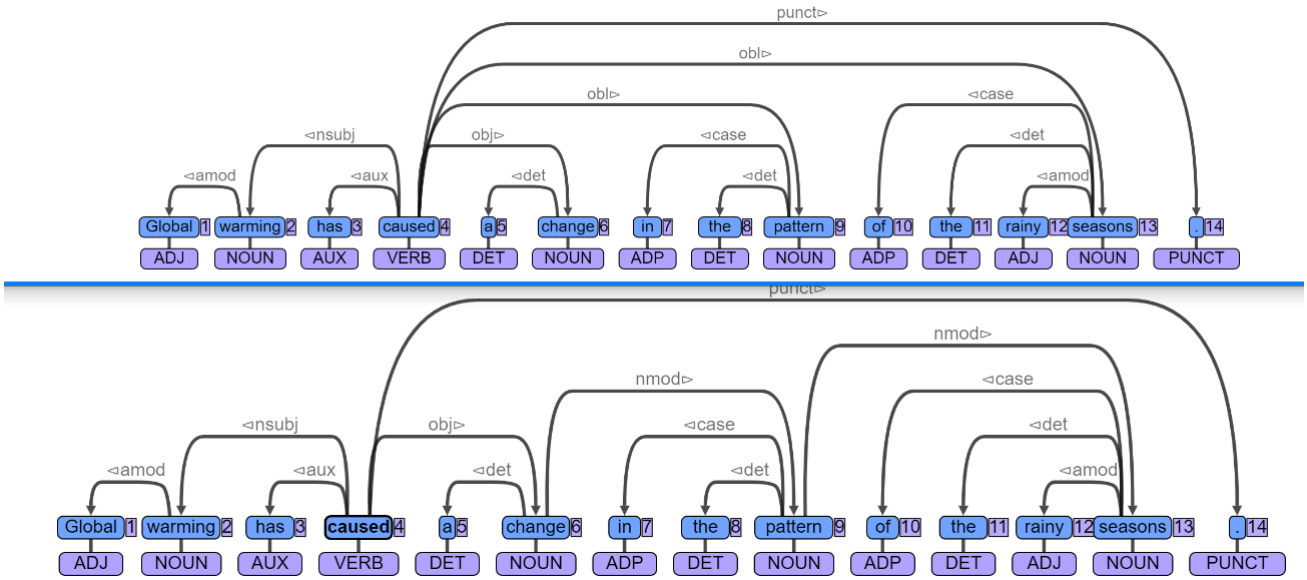
1

I had six types of differing dependency relations across eight examples of my annotation differing from the gold standard. I've decided to go over them all below. All following images, using the UD Annotatrix annotation tool, have the orientation of Top: My Annotation, Bottom: Gold Standard.



Sentence 2: ***"She spoke to CNN Style about the experience."***

- I had an oblique nominal (obl) dependency arc from the main predicate of "spoke" to "CNN" as an obl at first and in turn made "to" "CNN"'s case. "CNN" is "Style"'s compound while "Style" is the real obl and that would then make "to" attach to "Style" as the case.

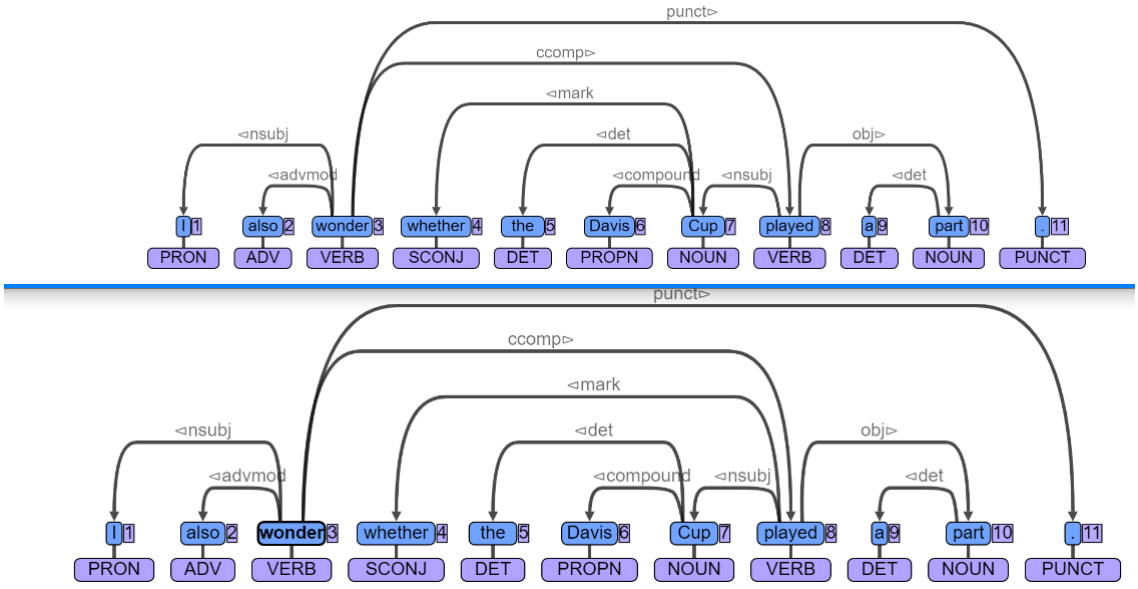


Sentence 3: ***"Global warming has caused a change in the pattern of the rainy seasons."***

- I originally had an oblique nominal (obl) dependency arc from "caused" to "pattern" thinking this relationship would function in a similar manner to the "spoke"-to-"experience" obl dependency arc in sentence 2. Really the main arc linking to pattern from the first half of the sentence is in the form of a nominal modifier (nmod) stemming from "change".

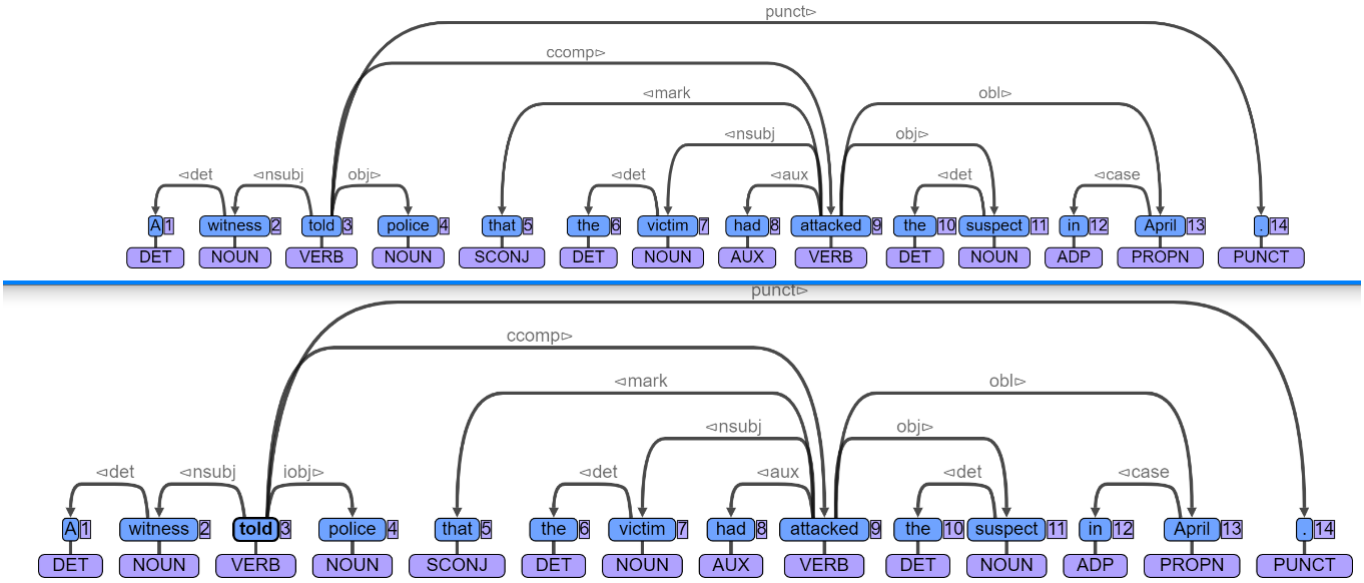
Sentence 3: ***"Global warming has caused a change in the pattern of the rainy seasons."***

- Due to the first issue in Sentence 3, an absence of a a nmod dependency arc from pattern to seasons immediately followed in my annotation.



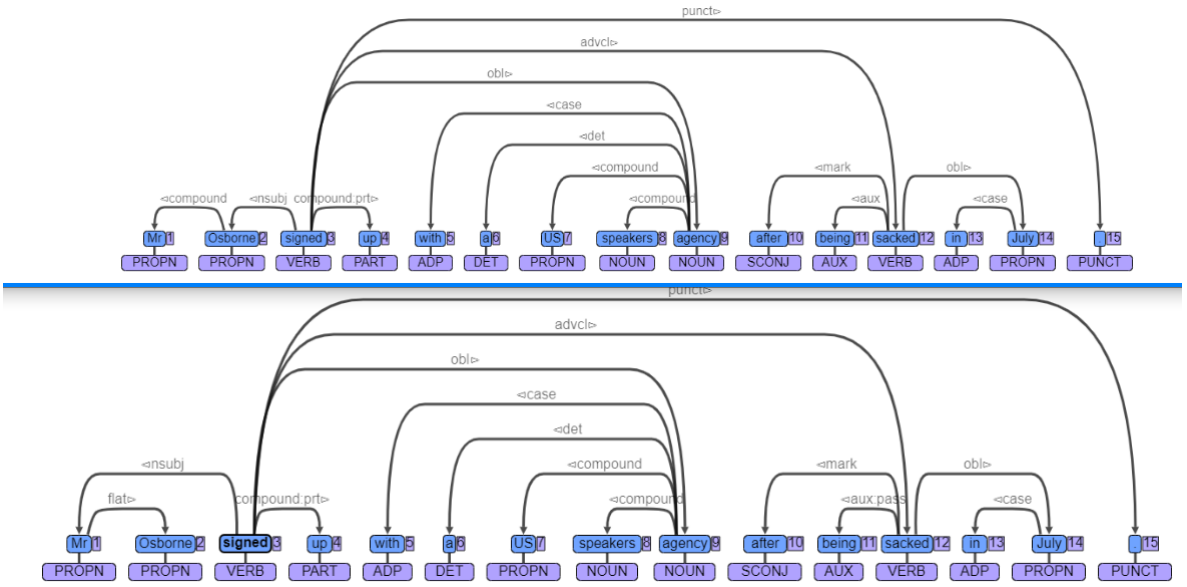
Sentence 4: ***"I also wonder whether the Davis Cup played a part."***

- I had a marker (mark) dependency arc from "Cup" to "whether", but this is incorrect as "whether" is being introduced to the complement clause "played".



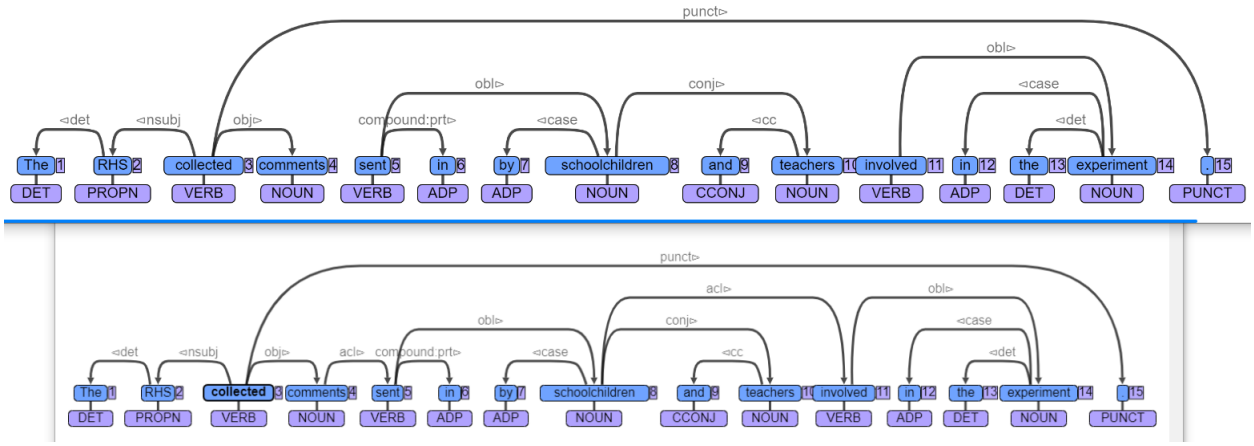
Sentence 7: ***"A witness told police that the victim had attacked the suspect in April."***

- I had an object (obj) dependency arc from the main predicate of "told" to "police", when it should've been an indirect object. The gold standard is correct in this as I overlooked "police" being the recipient of a ditransitive verb of exchange.



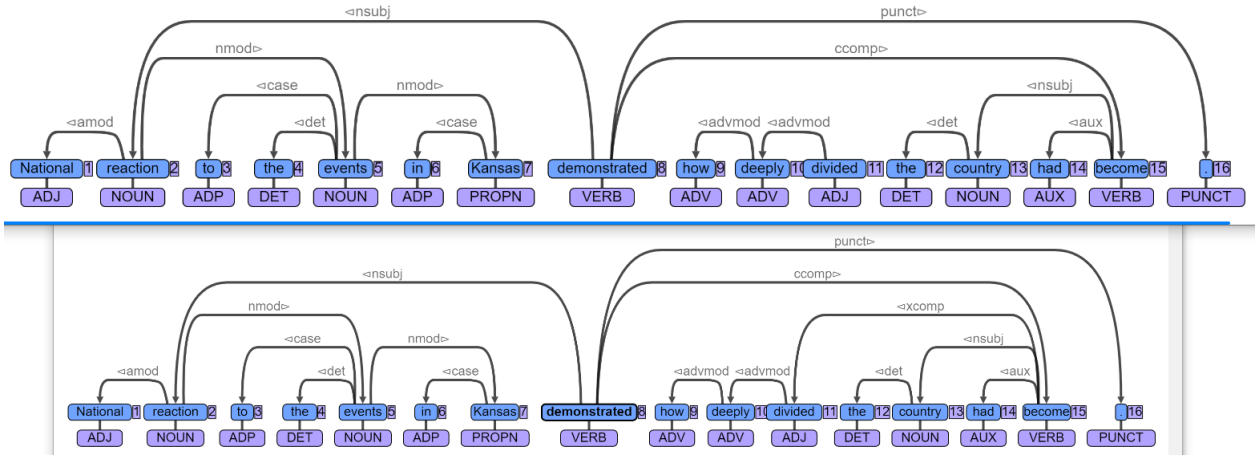
Sentence 8: ***"Mr Osborne signed up with a US speakers agency after being sacked in July."***

- I incorrectly arced "signed" to "Osborne" and creating a compound with the latter. The gold standard is correct in this being a flat multiword expression instead, as "Mr Osborne" is an headless/exocentric semi-fixed multiword expression and isn't an endocentric multiword expression. This is common for names. However, I was confused as to why the nsubj dependency arc from the main predicate "signed" went to "Mr" instead of "Osborne". The UD defines flats as "these expressions do not have any internal syntactic structure and that the structural annotation is in principle arbitrary", so I can assume that it's desirable to keep this hierarchy between main predicate and flat head consistent across language.



Sentence 9: ***"The RHS collected comments sent in by schoolchildren and teachers involved in the experiment."***

- I initially didn't include two clausal modifier of noun (acl) dependecy arcs between "comments"-and-"sent" and between "schoolchildren"-and-"involved". The standard is correct here because the head of the acl relation, the noun "comments", is being modified by the dependent of the arc, the head of the clause "sent". This also directly applies to the relationship of "schoolchildren" and "involved".



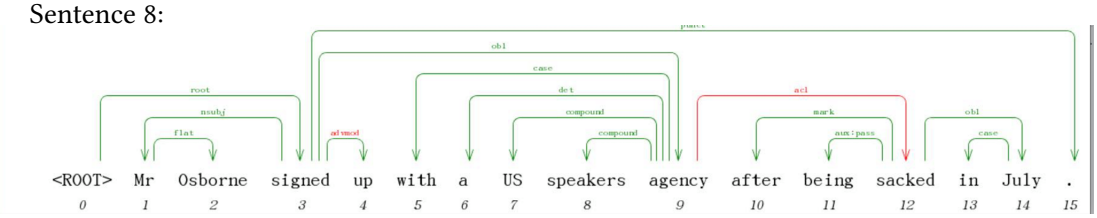
Sentence 10: ***"National reaction to the events in Kansas demonstrated how deeply divided the country had become."***

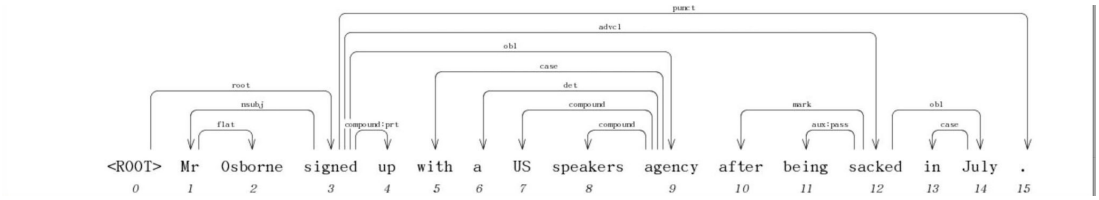
- I didn't include an open clausal complement (xcomp) from "become" to "divided" as is present in the gold standard. I agree that because the clausal complement doesn't have its own subject it can be seen as an xcomp. Additionally, because they are not modifying anything they are seen as non-finite which is another characteristic of such complements.

2

The below image has the orientation of Top: Annotated, Bottom: Gold Standard.

Sentence 8:





First Error: " agency sacked " -
From sentence 8, a clausal modifier of noun dependency arc (acl) was made by the parser between from "sacked" to "agency". This is in error because once you recognize "sacked" as the main predicate of the sentence, and that it is clausal, it cannot be assigned as a clause that modifies the nominal "agency". As "sacked" modifies the verb "signed" in the gold standard, you can see that this relationship is displayed as a modifier and not as a complement. However, I can see how it might have been incorrectly parsed given the second error also.

Second Error " signed up " -
Also from sentence 8, the relationship between "signed" and the following word "up", according to the gold standard, is that of a fixed pair of verb and adverbial particle (with the verb signed being in the form of 'sign'). Categorization of the sequence "sign up" was erroneously labeled as an adverbial modifier dependency arc (advmod). We might be able to surmise that the parser classified "up" as the predicate "sign"'s modifier, because once you've reviewed the Universal Dependency (UD) rule-set for word relations[1] you can see that they state that an adverbial modifier (advmod) is created once a non-clausal adverb or adverbial phrase modifies a modifier or predicate.

3

There are 3 important aspects used in determining the configuration of a given sentence by the arc-eager transition system used by the parser in Lab 9. These are a buffer of words in a sentence, a stack of said words, and a dependency graph at a given state. Transition-based parsing utilizes the history of the parsing to foresee the next transition state and will then match a specific input sentence to the best parse possible along with its dependency arcs/relations. A partial/working tree is created at each stage as the transition system works in a left-right orientation, going through the input sentence. The allocation of data will spill from the first element, the buffer, into the next, the stack, as the system progresses through the sentence. At the initial stage, all the words will be in the buffer and none in the empty state and nothing in the dependency graph. However, by the final stage the buffer will be the empty element and the stack will have a single word. Below are the transition sequences for a possible parse tree for sentence one.

[1] universaldependencies.org/u/dep/advmod.html

Table 1: Transition Sequences for a Parse Tree of Sentence 1

T	S	B	A
[]	[ROOT]	[He worked...]	[∅]
[SH]	[ROOT, He]	[worked for...]	[∅]
[LA]	[ROOT]	[worked for...]	$[worked_2, nsubj, He_1]$
[RA]	[ROOT, worked]	[for the...]	$[ROOT_0, root, worked_2]$
[SH]	[ROOT, worked, for]	[the BBC...]	[–]
[SH]	[ROOT, worked, for, the]	[BBC for...]	[–]
[LA]	[ROOT, worked, for]	[BBC for...]	$[BBC_5, det, the_4]$
[LA]	[ROOT, worked]	[BBC for...]	$[BBC_5, case, for_3]$
[RA]	[ROOT, worked, BBC]	[for a...]	$[worked_2, obl, BBC_5]$
[SH]	[ROOT, worked, BBC, for]	[a decade...]	[–]
[SH]	[ROOT, worked, BBC, for, a]	[decade .]	[–]
[LA]	[ROOT, worked, BBC, for]	[decade .]	$[decade_8, det, a_7]$
[LA]	[ROOT, worked, BBC]	[decade .]	$[decade_8, case, for_6]$
[RED]	[ROOT, worked]	[decade .]	[–]
[RA]	[ROOT, worked, decade]	[.]	$[worked_2, obl, decade_8]$
[RED]	[ROOT, worked]	[.]	[–]
[RA]	[ROOT, worked,.]	[∅]	$[worked_2, punct, .9]$

4

Directly below is the format of the grammar.txt used in Lab 10, only updated:

Grammar
S -> S Punct | Conj S
S -> NP VP
NP -> Pronoun | Det Noun | Noun Noun | NP PP | Adj Noun | Det NP | Noun Conj Noun | Uncountable-noun
VP -> Verb PP PP | VP NP | Aux Verb | Verb S | Adv VP | Verb NP
PP -> Prep NP

Lexicon
Noun -> 'BBC' | 'decade' | 'CNN' | 'Style' | 'experience' | 'warming' | 'change' | 'pattern' | 'seasons' | 'Davis' | 'Cup' | 'part' | 'scheme' | 'sponsorship' | 'advertising' Verb -> 'worked' | 'spoke' | 'caused' | 'wonder' | 'played' | 'makes'
Pronoun -> 'He' | 'She' | 'I'
Det -> 'the' | 'a' | 'The'
Prep -> 'for' | 'about' | 'to' | 'of' | 'through' | 'in'
Punct -> '.'
Aux -> 'has'
Adj -> 'rainy' | 'Global'
Adv -> 'also'
Conj -> 'whether' | 'and'
Uncountablenoun -> 'money'

Figures 1 and 2 below are the parse trees generated utilizing the grammar stated just above.

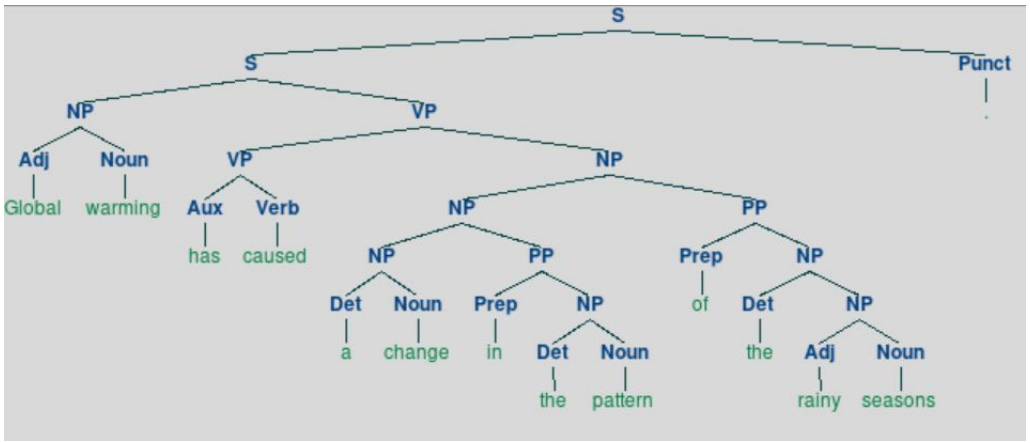


Figure 1: Tree 1

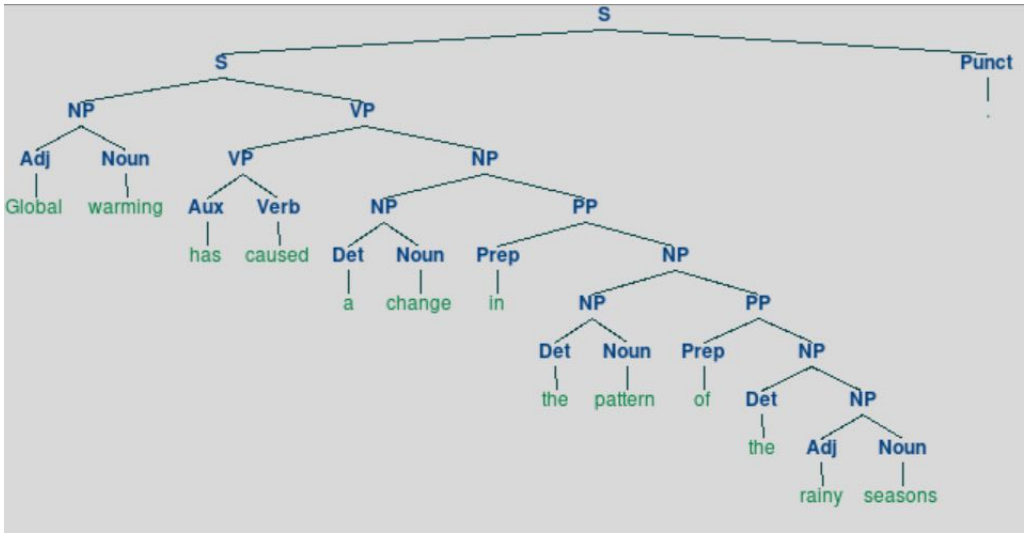


Figure 2: Tree 2

Figure 1's tree is dependent on the breaking up of "a change in the pattern of the rainy seasons" into NP and PP categories. The highest level NP (in the context of the excerpt previously stated) is subdivided into NP and PP, and the following PP subdivided into Preposition and NP categories. The small part of "a change" being broken up into NP and PP on the same branching level as "in" with "the pattern" makes me believe this branching method makes sense. This is because "A change in the pattern" can be considered an NP constituting of NP "a change" and PP "in the pattern." Although this isn't wrong, I preferred the a different representation that splits the relation from "the pattern of the rainy seasons." Figure 2's hierarchy displays what I think is a better and more easily understood reflection in dependencies within the phrase "a change in the pattern of the rainy seasons." It stems from how we classify "a change" in the subordinate NP, and "in the pattern" in the PP and how the branches split off after that. This differs from Figure 1's tree as this lays out a clearer depiction of "in the pattern of the rainy seasons" modifying the NP "a change" and how "in the pattern of the rainy seasons" should be considered a single phrase with the dependencies flowing down. Again, it's not that Figure 1 is incorrect; Figure 2 simply makes clearer sense to me the dependent relationships between the higher level NP and PP branches and what comes below them.

5

The CKY algorithm is a dynamic-programming solution to handle structural ambiguity when using parsing algorithms. Attachment and coordination ambiguity are the two problems, as outlined in our textbook by Jurafsky and Martin. Grammar assigning more than a single parse to a sentence is the main culprit of why this is such a problem. If you are given, or take, a context free text of grammar, you are not privy to the ability of parsers being able to determine which generated parse tree is more accurate, because of course - they are called "context free". NLP systems have to be able, or are at least very much desired to be able to, pick a single parse for any given sentence, but there is much ambiguity in the amount of text there is in the world and this will lead to a desperate need for syntactic disambiguation algorithms that can aid parsing algorithms in contextual, statistical and semantic data. The CKY is quite good because it's dynamic nature allows it to systematically generate parse trees once give an input. Additionally, grammars have to be inputted in the Chomsky Normal Form (CNF) format for CKY. You also have to ensure that your grammar doesn't mix up terminals and non-terminals, so you can create input rules to avoid this. Creating a "dummy non-terminal" that takes care of the original terminal is a must. This means that we change the terminals in our rules to dummy non-terminals but retain the rules that are compatible to Chomsky Normal Form (CNF). Then, convert unit productions and make sure all your rules are binary previous to importing them to the new grammar. Finally, review these steps and ensure that all non-terminal nodes in the parse tree break downward into 2 "offspring" above the part of the speech (POS). This allows us to have a tree put into a 2D matrix with each cell having an [i,j] format, the constituents ranging from the index of positions i to j, which was provided by the non-terminals where they come from. With each [i,j] we have, we'll also have a related position input "k", that splits off into 2. The first constituent [i,k] will be to the left of the [i,j] input and on row "i". The cells will be filled in bottom to top, left to right, so that all entries that add information to the currently worked on cell will already be on display elsewhere in the matrix. Left column are filled in first, then columns below since the algorithm works bottom to top. The left loops works on columns and the inner loop works on the rows. While the inner loop works, "k" traverses through the cells of [i,j] in left-right fashion for row "i" and in bottom-top fashion for row "j" for the potential spots a string could split off. Once a split is found by the algorithm it will then determine whether or not there's a potential combination of the two within the rules of the grammar. The non-terminal on the left of the rule will then be inputted into the matrix. It's important to remember that this isn't a parser, it's a recognizer. If you want to make it into a parser then you must cache away the current constituents that were put together to make every new constituents you inputted into the matrix. Adjustments must be made to the algorithm if you want to include probabilities of parses and overcome the issue of all the parses having the same level of priority.

0 He	**1 worked**	**2 for**	**3 the**	**4 BBC**	**5 for**	**6 a**	**7 decade**
1. Pron [0,1]	9.[0,2]	16.[0,3]	22.[0,4]	27.[0,5]	31.[0,6]	34.[0,7]	36.S[0,8]
	2.Verb [1,2]	10.[1,3]	17.[1,4]	23.[1,5]	28.[1,6]	32.[1,7]	35.VP[1,8]
		3.Prep[2,3]	11.[2,4]	18.PP[2,5]	24.[2,6]	29.[2,7]	33.[2,8]
			4.Det[3,4]	12.NP[3,5]	19.[3,6]	25.[3,7]	30.[3,8]
				5.Noun[4,5]	13.[4,6]	20.[4,7]	26.[4,8]
					6.Prep[5,6]	14.[5,7]	21.PP[5,8]
						7.Det[6,7]	15. NP[6,8]
							8.Noun[7,8]

Table2: A parse matrix utilizing the CKY algorithm

6 VG: Constituency vs. Dependency

Dependency is a syntactic parsing of natural language fueled by linguistic dependency grammar that comes somewhat from tradition. Constituency is a set collection of words that are organized as constituents. Each determining other elements. Phrase structure arranges words into embedded constituents, leading constituency to be see as a "part-whole relation." Even if we'd like to single out

constituents, you need to remember that they must be able to perform as a unit in differently placed positions. Dependency can be very advantageous for, say, Semitic languages like Amharic or Arabic which are morphologicaly rich. These languages have many, many terminal nodes stemming from their inflectional forms and dependency can utilize it's concentration of relationships between head and dependent elements to find success in this area. Additionally, if the language has loose/free word order and are not bound by stringent rule-sets or order, dependency would be a superior option over constituency. This also helps later down the line if it's desired to extract information for data extraction or relating back to previous text to answer a question. This is done by dependency easily drawing out principle subj - predicate - obj relations. It's very helpful to remember a key different between constituency and dependency the relationship between the head and dependent. Constituent structures become unneeded and redundant in dependency parsing because they're attached concretely, whereas there are numerous levels of constituents tying back to the main relation in constituent structures solely.

It's important to keep an open mind when answering which is better, constituency or dependency. It can depend upon the language being worked on. They both have parsers that are useful for different tasks within natural language processing. Dialogue understanding, grammar checking, and machine translation can be worked on more easily with context free grammars, while question answering, semantic parsing and data extraction can be worked on more easily with underlying dependency parsing stemming from a set of more formal grammar rules.