**Dependency Parsing**

**What is parsing?:**

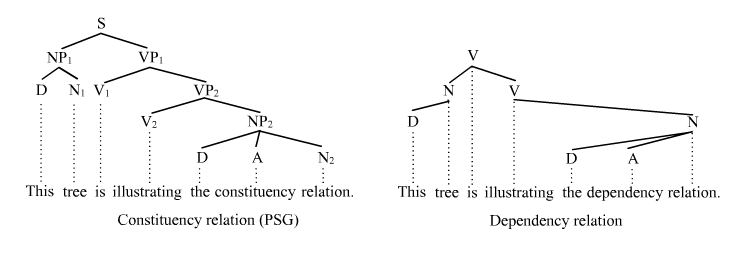
Parsing remains one of the most important processes we can carry out in the text. It isn't limited to natural languages though and has a history of computer languages as well.

To be able to do any kind of parsing, we would need two things – a parser and a grammar.

We can understand it is a way to analyze a sentence or breaking up a sentence to **understand** the **structure** of a sentence.

**Dependency parsing** refers to understanding the structure of a sentence via the dependencies between words in a sentence. Dependency is the idea that words in a sentence are connected to each other with directed links.

**Phrase structure parsing**, on the other hand, breaks up sentences into phrases, or separate constituents, and can also be referred to as constituency parsing. So, while a sentence that is dependency parsed would give us information about the relationships between words in a sentence, a sentence that is parsed using constituency relationships will help us understand how we can group our sentences.

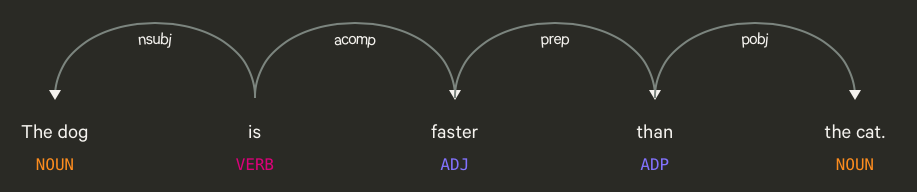


Constituency parsers break up a sentence into a subject and an object, which is usually a noun phrase and a verb phrase. Dependency parsers, on the other hand, consider the verb as the head of the sentence, and all dependencies are built around it.

Identifying the subject and the object as well, gives some semantic information about the context of words which might be previously unknown.

There are many kinds of dependencies which can be represented during such a parsing; the popular ones being:

* semantic dependencies,
* morphological dependencies
* prosodic dependencies
* **syntactic dependencies**

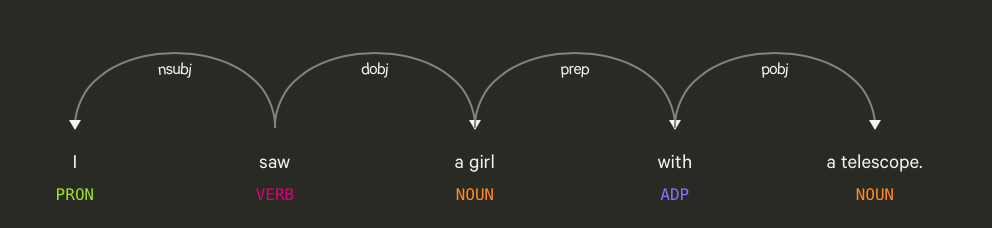


The root word is is, which is the main verb of the sentence. The dog is the noun phrase which is marked as nsubj, which refers to the nominal subject of the sentence. Acomp means adjectival complement, which means that it is a clause or phrase that modifies an adjective or adds to the meaning to an adjective. The word than is our preposition; and pobj stands for the object of a preposition, which is here the cat.

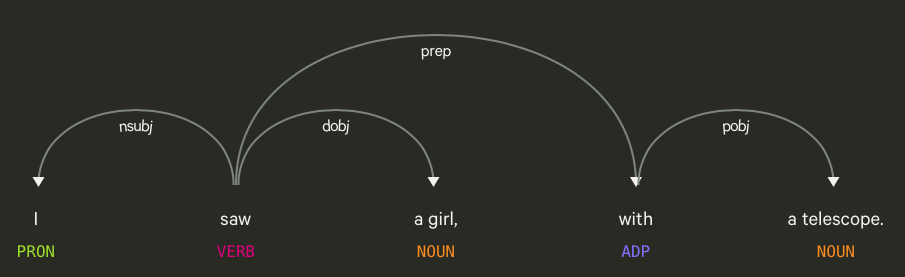
**Where is this information of phrases or of dependencies going to come in handy in text analysis?**

* Having a sentence parsed with phrasal rules can help us in NER-tagging.
* Extract a knowledge graph from the tree, which will give information regarding the words and how they relate to one another. Using such a knowledge graph as an intermediate step, we can attempt to perform language agnostic translation.
* This knowledge graph representation of a sentence can also be helpful when constructing chatbots or a system where we have to understand tasks that need to be performed - in this case, identifying actions are very important.
* Verify the grammatical correctness of a sentence.
* **Ambiguity**

I saw a girl with a telescope.



I saw a girl, with a telescope.



Dependencies helped resolve ambiguities between two very similar sentences.

**Dependency parsing in Python:**

spaCy's parsing portion of the pipeline does both phrasal parsing and dependency parsing - this means that we can get information about what the noun and verb chunks in a sentence are, as well as information about the dependencies between words.

Phrasal parsing can also be referred to as chunking, as we get chunks that are part of sentences, which are phrases. These chunks are stored in each sentence noun\_chunks attribute.

import spacy

nlp = spacy.load("en\_core\_web\_sm")

sent\_0 = nlp(u'Myriam saw Clement with a telescope.')

sent\_1 = nlp(u'Self-driving cars shift insurance liability toward manufacturers.')

sent\_2 = nlp(u'I shot the elephant in my pyjamas.')

for chunk in sent\_0.noun\_chunks:

    print(chunk.text, chunk.root.text, chunk.root.dep\_,

    chunk.root.head.text)

**(u'Myriam', u'Myriam', u'nsubj', u'saw')**

**(u'Clement', u'Clement', u'dobj', u'saw')**

**(u'a telescope', u'telescope', u'pobj', u'with')**

We can see here that we now have the chunks, the root text (we can see this in the a telescope chunk, whose root is telescope ), the dependency type, and the head. As expected, as the verb is saw, it is the head for both Myriam and Clement , where Myriam is the subject and Clement is the object.

for token in sent\_0:

    print(token.text, token.dep\_, token.head.text, token.head.pos\_,

    [child for child in token.children])

**(u'Myriam', u'nsubj', u'saw', u'VERB', [])**

**(u'saw', u'ROOT', u'saw', u'VERB', [Myriam, Clement, with, .])**

**(u'Clement', u'dobj', u'saw', u'VERB', [])**

**(u'with', u'prep', u'saw', u'VERB', [telescope])**

**(u'a', u'det', u'telescope', u'NOUN', [])**

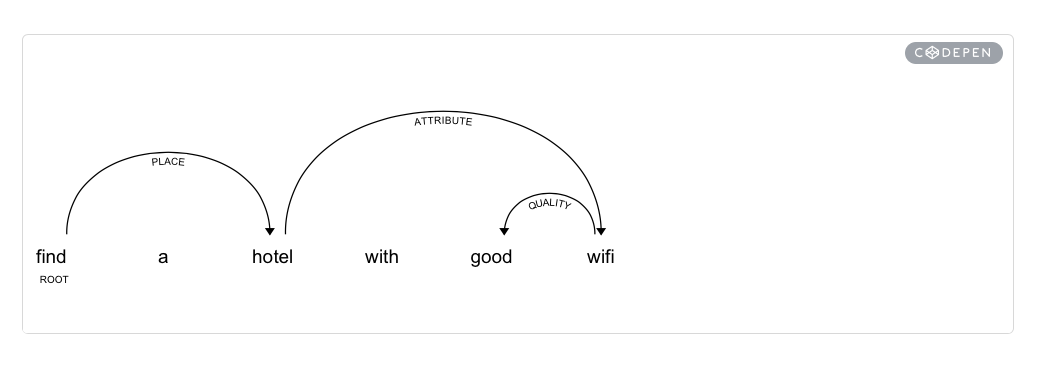
**(u'telescope', u'pobj', u'with', u'ADP', [a])**

**(u'.', u'punct', u'saw', u'VERB', [])**

The output is similar to the noun chunk examples, with the addition of a list that contains the children (if any) of the nodes. We can see immediately with the preceding example that the word saw , the root verb, is the head node, with four children nodes that are dependent on it, which is visible in the list.

**Training a dependency parser:**

We can now train our parsers to understand new semantic relationships or dependencies between words. The spaCy documentation page gives us the following example to illustrate this:



This is particularly interesting because we can model our own dependencies that are useful for our particular use-cases; though we must keep in mind that it may not always result in correct dependency parsing, but it is still useful in encapsulating relationships between words.

**Homework:**

Train a dependency parser to recognize a new dependency called “Quantity”. The adjectives that must be tagged as so are: **“few,” “many,” “many,” “enough,” “some,” “all,” “half,” “whole,” “enough,” and “numerous”.**

Your training data must include all these adjectives.