ANLP Tutorial Exercise Set 3 (for tutorial groups in week 6) WITH SOLUTIONS

v1.2

School of Informatics, University of Edinburgh
Henry Thompson, Sharon Goldwater

This week's tutorial exercises focus on syntax, (English) grammar, and parsing, using both context-free grammar and dependencies. After working through the exercises and discussing some additional issues in your tutorial groups, you should be able to:

- Provide examples showing how syntactic structure reflects the semantics of a sentence, and in particular, semantic ambiguity. You should also be able to explain and illustrate how constituency parses and dependency parses differ with respect to this issue.
- Provide the syntactic parse for simple sentences using either Universal Dependencies or context-free grammar rules.
- Hand-simulate the CKY parsing algorithm and transition-based dependency parsing, and by doing so, better understand some of the computational issues involved.

1 CFGs and attachment ambiguity

When constructing a grammar and parsing with it, one important goal is to accurately reflect the meaning of sentences in the structure of the trees the grammar assigns to them. Assuming a compositional semantics, this means we would expect attachment ambiguity in the grammar to reflect alternative interpretations. The following two exercises aim to hone your intuitions about the syntax-semantics relationship.

Exercise 1.

In English, conjunctions often create attachment ambiguity, as in the sentence `I like green eggs and ham`. The ambiguity inside the noun phrase here could be captured by the following two context-free grammar rules, where `Nom` is a nominal (noun-like) category:

`Nom` → `Adj Nom`
`Nom` → `Nom Conj Nom`

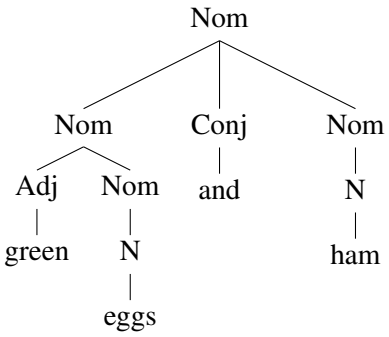
a) Write down two paraphrases of `I like green eggs and ham`, where each paraphrase unambiguously expresses one of the meanings.

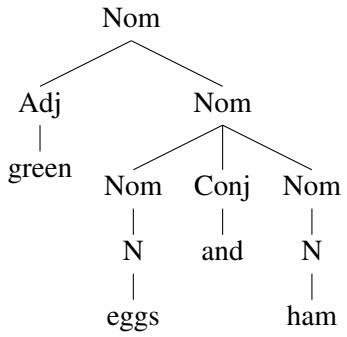
b) Draw two parse trees for just the `Nom` part of the sentence, illustrating the ambiguity. You'll also need to use a rule `Nom` → `N`. Which tree goes with which paraphrase?

Solution 1.

a) `I like ham and green eggs` or `I like green eggs and green ham`.

b) Respectively:





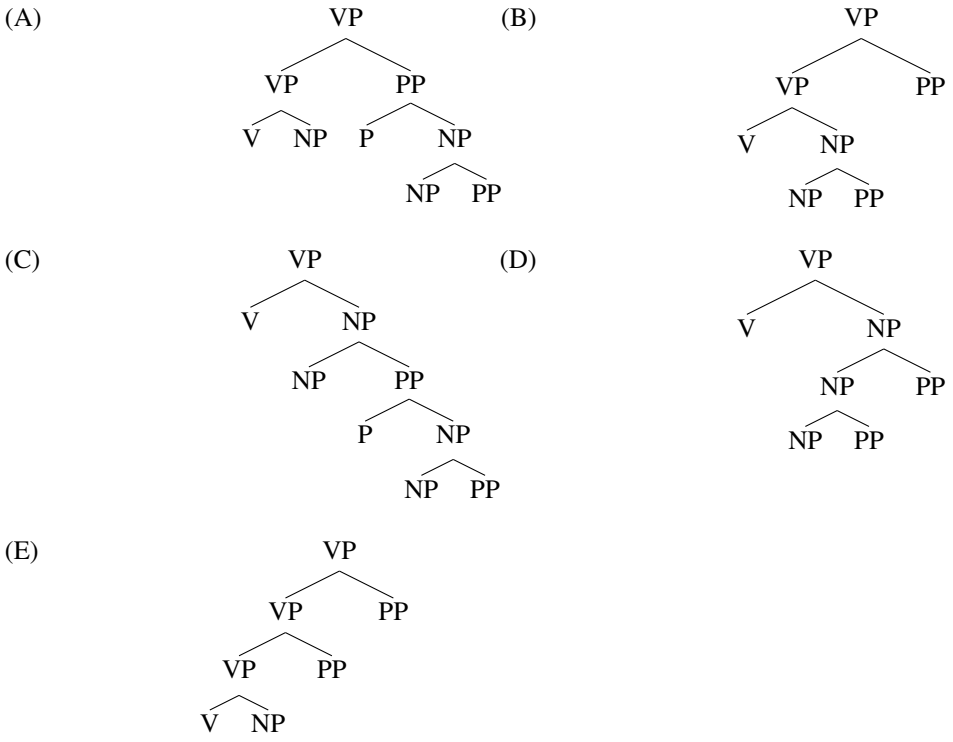


Figure 1: Trees for exercise 2

Exercise 2.
Another common source of attachment ambiguity in English is from prepositional phrases. The relevant grammar rules include:

VP → V NP
VP → VP PP
NP → NP PP
PP → P NP

Here are five verb phrases:

(1) watched the comet from the roof with my telescope
(2) watched the comet from the park across the street
(3) watched a video by the BBC about the comet
(4) watched a video about the expedition to the comet
(5) watched a video about the comment on my mobile

Figure 1 shows five partial trees. Match the phrases to the trees which best capture their meanings. You may find it helpful to ask yourself questions such as "where did this event happen?", "how was it done?", "what was watched?". You may also want to try out (in pencil!) different ways of writing in phrases under the leaves of the various trees.

Solution 2.
1E; 2A; 3D; 4C; 5B

2 CKY parsing

Exercise 3.
Assume we are using the following grammar:

S → NP VP	V → swam | ran | flew
VP → V NP	VP → swam | ran | flew
VP → VP PP	D → the | a | an
NP → D N	N → pilot | plane
NP → NP PP	NP → Edinburgh | Glasgow
PP → P NP	P → to

a) Draw a 7x7 chart for the sentence `the pilot flew the plane to Glasgow` and fill it in using the CKY algorithm. Number the symbols you put in the matrix in the order they would be computed, assuming the grammar is searched top-to-bottom.

b) How is the attachment ambiguity present in this sentence reflected in the chart at the end?

Solution 3.

a) Here is a picture of the chart. To avoid clutter I included the backpointers only for the final three items added (the VPs and S). The backpointers show the (i,j) indices for the pair of child cells.

	1 the	2 pilot	3 flew	4 the	5 plane	6 to	7 Glasgow
0	1:D	9:NP	12:S		15:S		18:S [NP(0,2),VP(2,7)]
1		2:N					
2			3:V 4:VP		13:VP		16:VP [V(2,3),NP(3,7)]
							17:VP [VP(2,5),PP(5,7)]
3				5:D	10:NP		14:NP
4					6:N		
5						7:P	11:PP
6							8:NP

b) The ambiguity isn't represented explicitly at the top node. However if we follow the backpointer, we see that there are two VPs in (2,7), which indicates two distinct subtrees (with different backpointers).

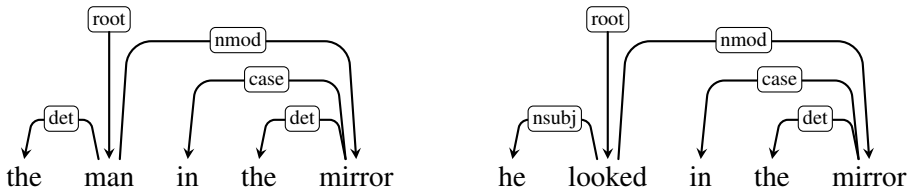
It's actually important that we do **not** add a second S at the top: if we carried the ambiguity upward in this fashion, we could end up storing an exponential number of categories in each cell—and this is exactly what we are trying to avoid.

Notice that the backpointers contain both the label and location of each child. For this example, the location alone would be enough because (for example) there is only one way to build a VP from the items in (2,5) and (5,7). But in principle there might be more than one rule that can make a VP from items in those cells, and in order to be able to efficiently reconstruct all of them at the end, we need to know the child's label as well as its location. What we end up with at the end of parsing is called a "packed parse forest."

3 Dependency syntax and parsing

Exercise 4.

a) Draw dependency parses for verb phrases (2) and (3) from exercise 2, using UD labels for the relations as illustrated in JM3. You shouldn't need to know any more labels than: `nsubj`, `dobj`, `iobj`, `det`, `case`, `nmod`, `amod`. You should be able to figure out most of the labels by looking at examples from the textbook. For prepositional phrases, use the `nmod` relation, as in these examples:[1]

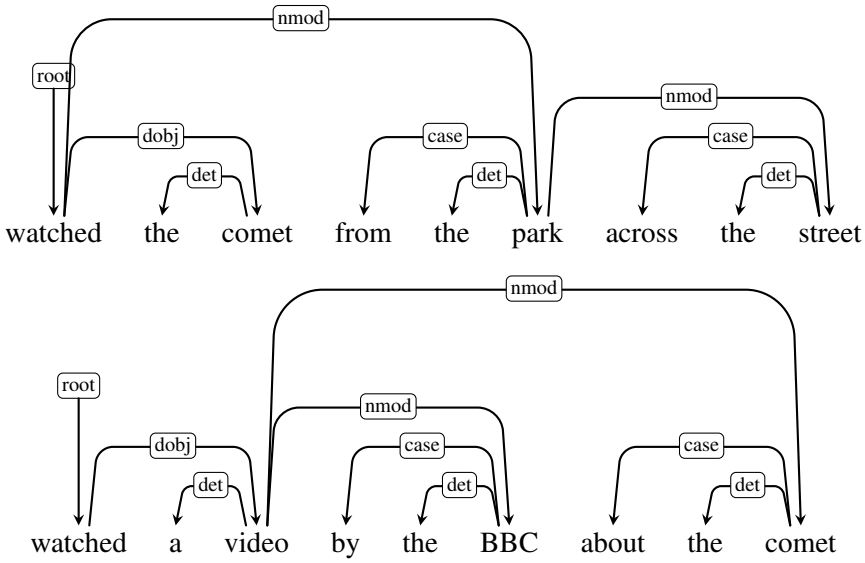


Note that by convention, all dependency parses have a root, whether or not it is the head of a full sentence. Also note that there's a mistake in JM3 figures 13.5, 13.6, and 13.15 where the (book → me) relation should be labelled `iobj` and the (book → flight) relation should be labelled `dobj`.

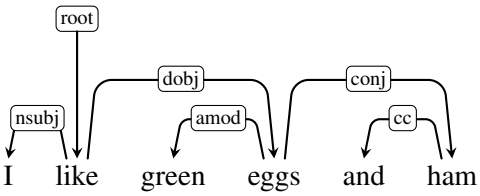
b) Now try to draw a dependency parse for the sentence `I like green eggs and ham`. You will need to use the `cc` and `conj` labels (see examples in JM3, Fig 13.3). Do you run across any problems? Is it clear what the dependency structure should be? Is the ambiguity in this sentence represented in the dependency structure (or multiple structures), and if so how?

Solution 4.

a) The two trees are:



b) The correct tree according to UD v1 guidelines (and following the textbook) is:



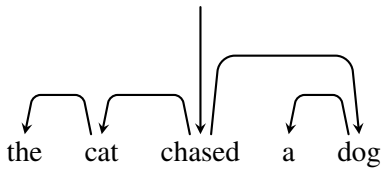
[1] The textbook and this tutorial more or less follow the UD v1 guidelines; UD guidelines have now been updated for v2, so if you look online you may find discrepancies.

There are several points worth noticing/discussing, including the following:

- Conjunction is inherently a symmetric relationship, but dependency grammar requires asymmetric relations. So it isn't a natural fit, and requires choosing one of the two conjuncts arbitrarily as the head for both the `conj` and `cc` relations. In fact UD v1 and v2 differ on what is the head of the `cc` relation!
- There is only a single dependency parse for this sentence, even though there are two different meanings. So in this case (unlike constituency structure) the dependency structure does not reflect the semantic ambiguity.
- You might have considered (or discussed in your group) whether there are alternative guidelines for parsing conjunctions that would reflect the ambiguity, or have a more symmetric relationship. For example, would it be reasonable to make *and* the head of the conjoined phrase, and would this solve any of the problems? What might a dependency-like structure look like that better captures the meaning where *green* modifies both *eggs* and *ham*? (It has two arcs pointing to *green*, which isn't a valid dependency tree. But some people have proposed that we should really be using dependency *graphs*, rather than *trees*. These would permit this kind of structure, but are much more difficult to deal with computationally, e.g. to design efficient parsing algorithms.)

The main takeaway from this exercise is for you to understand some of the weaknesses of dependency structure.

Exercise 5.
Consider the following dependency-annotated sentence. (For simplicity, we leave out the relation labels in this exercise).



By hand-simulating the algorithm for arc-standard transition-based parsing, show that there is more than one sequence of transitions that can lead to the correct parse of this sentence. How does this fact motivate the need for the procedure described in JM3 section 13.4.1 (generating the training oracle)? What is the sequence produced by the training oracle?

Solution 5.
Here are two possible sequences (you might find others). The first is the training oracle sequence, which chooses LEFTARC as soon as possible in all cases.

Step	Stack	Word list	Action	Relation added
0	[root]	[the, cat, chased, a, dog]	SHIFT	
1	[root, the]	[cat, chased, a, dog]	SHIFT	
2	[root, the, cat]	[chased, a, dog]	LEFTARC	(the ← cat)
3	[root, cat]	[chased, a, dog]	SHIFT	
4	[root, cat, chased]	[a, dog]	LEFTARC	(cat ← chased)
5	[root, chased]	[a, dog]	SHIFT	
6	[root, chased, a]	[dog]	SHIFT	
7	[root, chased, a, dog]	[]	LEFTARC	(a ← dog)
8	[root, chased, dog]	[]	RIGHTARC	(chased → dog)
9	[root, chased]	[]	RIGHTARC	(root → chased)
10	[root]	[]	DONE	

Step	Stack	Word list	Action	Relation added
0	[root]	[the, cat, chased, a, dog]	SHIFT	
1	[root, the]	[cat, chased, a, dog]	SHIFT	
2	[root, the, cat]	[chased, a, dog]	LEFTARC	(the ← cat)
3	[root, cat]	[chased, a, dog]	SHIFT	
4	[root, cat, chased]	[a, dog]	SHIFT	
5	[root, cat, chased, a]	[dog]	SHIFT	
6	[root, cat, chased, a, dog]	[]	LEFTARC	(a ← dog)
7	[root, cat, chased, dog]	[]	RIGHTARC	(chased → dog)
8	[root, cat, chased]	[]	LEFTARC	(cat ← chased)
9	[root, chased]	[]	RIGHTARC	(root → chased)
10	[root]	[]	DONE	

The training oracle is needed in order to define a set of actions that will lead to a correct parse, and are also as consistent as possible. In other words, when we train the classifier to decide an action, we want the training data (sequences of configurations from the training oracle) to be as consistent as possible about what action is taken given a particular configuration or partial configuration, because consistent patterns are easier to learn than random ones.