HW 5: Har two loo cat

Re-submit Assignment

Due May 28 by 11:59pm **Points** 100 **Submitting** a file upload

What you need to turn in:

- hw5part1.pl (see part 1)
- hw5part2.pl (see part 2)
- hw5.pdf (see parts 1, 2 and 3)

PART 1: Duck at in the garden

Unlike FSTs, prolog has built in support for CFGs, which makes them very easy to work with in this language. (For those who are curious, the prolog tutorial linked from the course front page on canvas has a section on Definite Clause Grammars (DCGs), which is how prolog impliments CFGs: click here (http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse29) for a link.) Because this feature is built in, to get started all we need to know is that if we have a context-free grammar rule like

$$S \longrightarrow NP\ VP$$

(where S stands for sentence, NP stands for noun phrase, and VP stands for verb phrase), we can write the rule in a prolog file as follows:

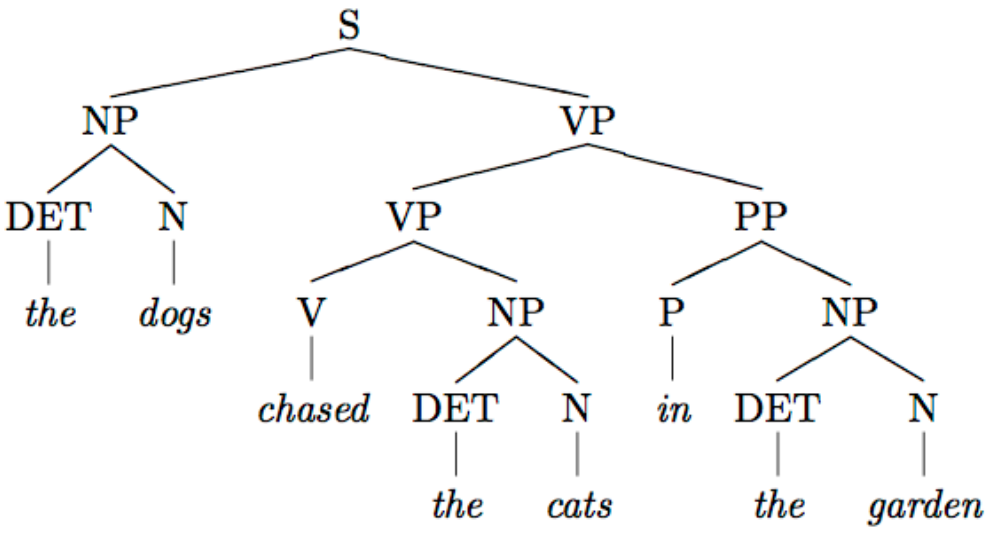
```
s --> np, vp.
```

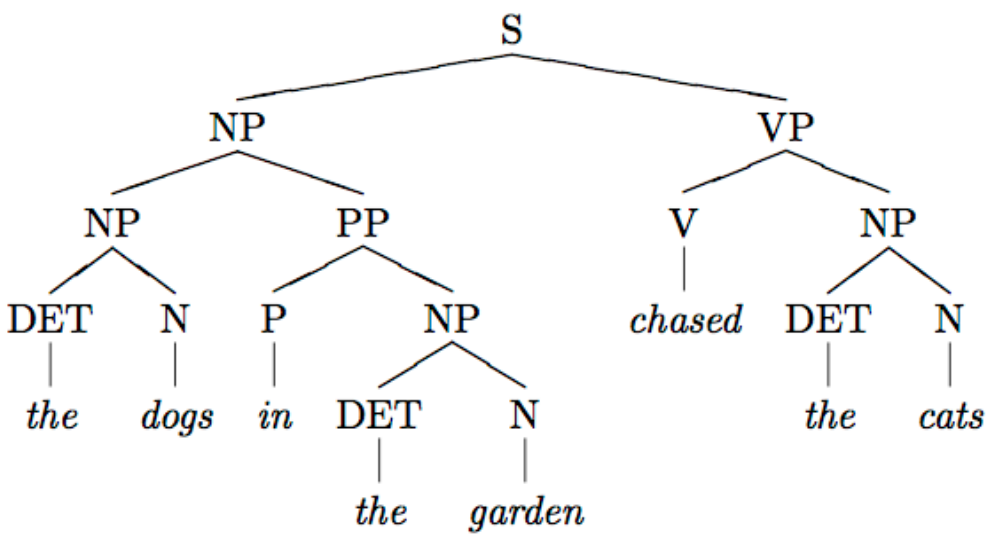
Note the period at the end, the comma between the symbols in the right-hand side, and the two dashes that form the arrow. The arrow must look exactly like the one above, there must always be a period at the end, and there must be comma between the symbols that appear on the right-hand side of the rule. Any deviation from this format with result in catastrophic failure (i.e., an error message).

However, this kind of rule only deals with nonterminals. Rules with terminals are similar, but you have to wrap the terminal in square brackets, as shown below:

```
n --> [dogs].
```

The first question regards the small treebank below:





Extract the context-free grammar (CFG) from this treebank, and implement it in prolog. In other words, write all of the rules necessary to create the trees above in a prolog file, following the format described above. Make sure to use lower case for the non-terminals, because otherwise prolog will think they are variables and the program will go haywire.

Hint: Yes, this is exactly as easy as it sounds. Just be careful, and check your work.

For example, notice that both trees need the rule

```
s --> np, vp.
```

So include that rule in your file. The add the rules that involve NPs, VPs, etc.

I've uploaded a partially complete solution in a prolog file named hw5part1-template.pl. You can find it in the HW5 folder on canvas. An important detail are the table statements at the beginning of the file. Make sure that all of the non-terminals that appear on the left-hand side of at least one rule in your CFG are listed there.

Once you have your grammar (name it hw5part1.pl) and it is in your working directory, you can test it by trying the following:

```
?- [hw5part1].
?- s([the,dogs,chased,the,cats], []).
```

which should return TRUE, and

```
?- s([the, dogs, the, cats, chased], []).
```

which should return FALSE.

You can also try

```
?- s(X, []).
```

and see what happens. (Do you know why? Answering this question is NOT a required part of the homework, but it is fun to think about.)

What to turn in for Part 1: a file called hw5part1.pl that contains your CFG *(do yourself a favor and don't turn it in until you've tested it)*. You don't need to turn in prolog output, just your grammar as a .pl file.

Part 2: Duck at and the trees

Now that we know how to use prolog to determine whether a given string is in the language defined by a CFG, let's see how to get prolog to return the parse trees for those strings. To get parse trees from prolog , we can add an argument to each non-terminal. The arguments of the non-terminals on the left-hand side of the rules tell us what subtree is created by that rule. These arguments need to be variables, because they need to match the values of the nonterminals on the right-hand side of the rule. Therefore, the arguments that appear on the left-hand side must also appear somewhere on the right-hand side. This is the same format that we saw in section. For example, the s rule above becomes:

```
s(s(NP, VP)) --> np(NP), vp(VP).
```

which is the same as saying

```
s(tree(Subtree1, Subtree2)) --> np(Subtree1), vp(Subtree2).
```

Here we see that the non-terminal symbol on the left-hand side is s. The argument of s tells us that by applying this rule we are forming a subtree with s at the root, and that this root has two children. Furthermore, each of these children is a subtree that we get from the two items on the right-hand side of the rule. Subtree1 is the subtree we get from the np, and Subtree2 is the subtree we get from the vp. Because the np and the vp are both phrases, the value of Subtree1 and Subtree2 will be whatever the content of those phrases is.

To complete Part 2, you will need to modify the grammar from part 1 so that it will produce trees by using rules with the format described above.

Download hw5part2-template.pl to get started with part 2 (it's in the HW5 folder). Again, notice the table statements, and make sure all of the phrases you add are listed.

Once your grammar is complete, try the following *(NOTE: this code is slightly different than the code for part 1 about -- now s is an s/3 predicate)*:

```
?- s(T, [the, dogs, chased, the, cats], []).
```

which should give you the parse tree as a value of T, and

```
?- s(T, [the, dogs, chased, the, cats, in, the, garden], []).
```

which should give you two different values of T, each of which is a valid parse tree according to the grammar. (After getting the first value of T, press semicolon to get the next value. If you don't get two values, you need to doublecheck your grammar.)

What to turn in for Part 2: a file called hw5part2.pl that contains a grammar which computes the trees described above. You don't need to turn in prolog output, just your grammar as a .pl file.

Part 3: The garden behind the house in the garden

Use your grammar from part 2 to parse the following sentences:

```
the dogs chased the cats
```

```
the dogs chased the cats in the garden
```

```
the dogs chased the cats in the garden by the house
```

```
the dogs chased the cats in the garden by the house by the garden
```

```
the dogs chased the cats in the garden by the house by the garden by the house
```

Notice that the number of possible trees increases rapidly as you increase the number of prepositional phrases (PPs). To see just how many parse trees there are for a sentence, use the following command:

```
?- findall(T, s(T, [the,dogs,chased,the,cats,in,the,garden], []), L), length(L, N).
```

The findall/3 predicate gives us all of the solutions for a variable (in this case, T), and puts them in a list (in this case, L). Then, the length/2 predicate tells us the number of elements there are in a list. In the example above, N is the number of elements in L. Because this predicate is built-in, you don't have to worry about N being a singleton: in this case, prolog knows how to handle it.

Create a table that shows the number of trees the CFG produces when we have 0, 1, 2, 3 and 4 PPs in a VP (as in the example sentences above). Can you identify the sequence that relates the number of PPs and the number of trees? (It is a known, named mathematical sequence, which can easily be found by searching for the first few numbers on google.)

What to turn in for part 3: a PDF report named hw5.pdf and enter the following:

- **The table described above, and the name (or a description) of the sequence that corresponds to the number of possible trees for sentences of increasing length.**