HW 4: Che's duck at

Submit Assignment

Due Monday by 11:59pm **Points** 100 **Submitting** a file upload
File Types pdf and pl

What to turn in

- hw4part1.pl (see part 1)
- hw4part2.pl (see part 2)
- hw4.pdf

PART 1: Duck at in the garden

Prolog gives us a simple way to write context-free grammars, thanks to built-in support for Definite Clause Grammars (DCGs). In this first part, we don't need to worry about what exactly DCGs are. (But for those who are curious, here's a link with a brief introduction: **http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse29** (http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse29) .) All we need to know is that if we have a context-free grammar rule like

S → 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 this way:

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

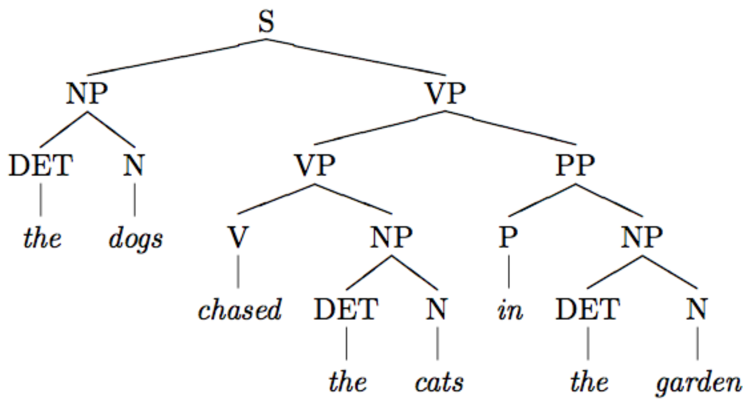
Note the period at the end, the comma between the symbols in the right-hand side, and the two dashes that form the arrow. The arrow must look like the one above, there must always be a period at the end, and there must be comma between symbols on the right-hand side.

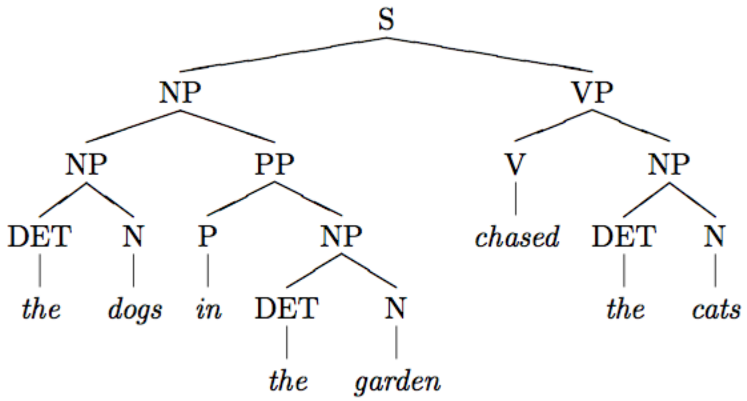
In class, we will see the difference between terminal symbols (used to form strings in the language) and non-terminal symbols (used in internal structure). The rule above deals only with non-terminals. Rules with terminal symbols are similar, but we will use square brackets around the terminal symbols, as in the example below:

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

Consider the small treebank below:

(A treebank is a collection of trees, where the trees are syntactic analyses for sentences.)





Extract the context-free grammar (CFG) from this treebank, and implement it as a prolog DCG. In other words, write all of the rules necessary to create the trees above in a prolog file, following the format described above. Use lower case for the non-terminals. (Again, all you need to know about DCGs for this part is that you can write the CFG rules as above.)

For example, notice that both trees need the rule

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

So include that rule in your file. Then add the rules that involve NPs, VPs, etc. Include each rule only once in your file. The rules that map parts-of-speech to English words (sometimes called the lexical rules) involve terminal symbols, so they will look like:

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

Download hw4part1-template.pl from the HW4 folder in canvas to get started. Notice the table statements in the beginning of the file. Make sure that all of our non-terminals are listed there.

Once you have your grammar (name it hw4part1.pl) and it is in your working directory, you can try the following:

```
?- [hw4part1].
?- s([the,dogs,chased,the,cats], []).
```

which should be true, and

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

which should be false.

TIP: When working on your assignment, after changing your grammar and reloading it in prolog, use the following to clear all tables computed for the previous grammar:

```
?- abolish_all_tables.
```

TURN IN: a file called hw4part1.pl with your grammar.

Part 2: Duck at and the trees

Now that we have seen how to use DCGs to determine whether strings are in the language defined by a CFG, let's see how to get the corresponding parse trees. To get parse trees from prolog DCGs, we can add an argument to each non-terminal that corresponds to the tree. The arguments of the non-terminals on the left-hand side of the rules tell us what subtree is created by that rule. For example, the s rule above becomes:

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

which is the same as

```
s(s(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, with two children, and 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.

Modify the grammar from part 1 to produce trees by using rules with the format above.

Download hw4part2-template.pl to get started with part 2. Again, notice the table statements, and make sure all of the non-terminals are listed.

Once your grammar is complete, you can try the following:

```
?- 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, with each being a valid parse tree according to your grammar. (After getting a value of T, press semicolon to get the next value.)

(You don't need to turn in prolog output, just your grammar.)

TURN IN: a file called hw4part2.pl with your grammar that computes trees.

Part 3: The garden behind the house in the garden

Starting with your grammar for part 2, add additional lexical rules for the following words: garden (n), fence (n), and behind (p). Use the resulting grammar 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 behind the fence
```

```
the dogs chased the cats in the garden behind the fence in the garden
```

```
the dogs chased the cats in the garden behind the fence in the garden behind the fence
```

Notice that the number of possible trees increases rapidly with the number of prepositional phrases (PPs). To see how many parse trees there are for a sentence, try:

```
?- 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 in a list. In the example above, N is the number of elements in L.

Create a table that shows the number of trees 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 sequence, which can easily be found by typing the numbers on google.)

TURN IN: Create a PDF report named hw4.pdf and enter the following:

- **The grammar for part 1 (ok to copy and paste from your .pl grammar)**
- **The grammar for part 2 (ok to copy and paste from your .pl grammar)**
- **Your table for part 3, and the name (or a description) of the sequence that corresponds to the number of possible trees for sentences of increasing length.**