HW 5: Don't arrive the dog

Submit Assignment

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

Part 1: Arrived the dog

Consider the CFG below, implemented in prolog:

```
s --> np, vp.
np --> det, n.
np --> pro.
vp --> v.
vp --> v, np.

det --> [the].
n --> [dog].
n --> [dogs].
pro --> [she].
pro --> [her].
pro --> [they].
pro --> [them].
pro --> [i].
pro --> [we].
v --> [arrived].
v --> [like].
v --> [likes].
```

This grammar accepts simple sentences like

```
she likes the dog
```

and

```
the dog arrived
```

It also accepts ungrammatical English sentences like

```
her likes them
```

and

```
they arrived the dog
```

In fact, we can check all of the sentences in the language defined by this grammar:

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

S = [the, dog, arrived]
S = [the, dog, like]
S = [the, dog, likes]
S = [the, dog, arrived, the, dog]
S = [the, dog, arrived, she]
...
```

And the list goes on.

In part 1 you will refine this grammar so that only grammatical English sentences are accepted. This will involve:

- Adding case to the noun phrases, and enforcing nominative case for subject and accusative case for objects. This should rule out *Her arrived*, while allowing *She arrived*. In this example, *she* is nominative, and *her* is accusative.
- Adding person and number information to verbs and nominals, and enforcing subject-verb agreement. This should rule out *She see the dog,* while allowing *She sees the dog*, and *They see the dog*. Notice that, in English, the subject and the verb must agree in person and number.
- Adding subcategorization information to verbs, marking them as transitive or intransitive. This should rule out *She arrived them*.

As a starting point, two files are provided:

- hw5part1plain.pl: the grammar shown above
- hw5part1case.pl: a modified grammar that takes care of the first bullet point above.

Inspect hw5part1case.pl carefully, comparing it to hw5part1plain.pl. What changed?

- In the lexicon, the pronouns that all had the category pro were split into two categories: pro_nom (pronouns with nominative case), and pro_acc (pronouns with accusative case).
- The noun phrase rules that used pronouns (with category pro) were changed to take case into account, by using either pro_nom or pro_acc.
- The non-terminal NP was replaced with two new non-terminals: NP_nom and NP_acc. The NP rules were changed accordingly: with pronouns, the case is simply passed on to the noun phrase, and with determiner noun, the noun phrase can have either case.
- Now that our NPs have case, the VP rule for transitive verbs was changed, so that the object of a verb must have accusative case (i.e. only NP_acc can be an object, not NP_nom).
- Similarly, the S rule was changed to that only NP_nom can be a subject.

Your task: now that case has been taken care of, modify the grammar further to account for subject-verb agreement and verb transitivity. Name your modified grammar hw5part1.pl.

Your modifications will likely involve changes to the lexical rules (splitting existing word categories into multiple categories, as we did with pro to account for case) and changing the phrase structure rules to

use the new word categories.

Your grammar should only generate grammatical English sentences, and it should be *principled* about how ungrammatical English sentences are ruled out. The goal is not simply to produce the correct set of sentences, but to create a grammar that is *accurate*, *simple* and *principled*.

Hint: If we think of what hw5part1plain.pl does, there are three different things to work on: case, subject-verb agreement, and verb subcategorization. The first one is done: hw5part1case.pl. If all of the changes together seem too confusing to keep track of, do agreement and transitivity, *one at a time*. These are separate issues. Then put it all together.

What to turn in for part 1: hw5part1.pl (your modified grammar).

Part 2: The features arrived the grammar

Splitting categories and non-terminals gets out of hand fast. An alternative to the proliferation of non-terminals is to attach features to the terminals and non-terminals, and use these features to constrain the grammar.

Consider the case example in part 1. We split pro into pro_nom and pro_acc, and we split np into np_nom and np_acc. The sequence *det n* could be either, so we added two rules (one for np_nom, and one for np_acc) with identical right-hand sides. Instead of splitting the symbols of the grammar, we can modify the lexical entries as follows:

```
pro --> [she].
pro --> [her].
```

could become

```
pro(nom) --> [she].
pro(acc) --> [her].
```

So far this is not so different from splitting pro into pro_nom and pro_acc. However, the difference is more apparent when we modify the NP grammar rules:

```
np_nom --> pro_nom.
np_acc --> pro_acc.
np_nom --> det, n.
np_acc --> det, n.
```

now become

```
np(C) --> pro(C).
np(_) --> det, n.
```

In the first rule, the np inherits its case feature C from the pro, and the pro gets it from the lexicon. Recall that we already changed the lexicon, so for the word she, we would have pro(nom), which can make an np(nom). In the second rule, instead of using an ordinary prolog variable, we used the underscore, which is called the anonymous variable; it matches anything, so we are just leaving the case feature unspecified (it can be set to something else in another rule).

In this simple example we see that instead of expanding two rules into four, we keep the number of rules intact, but still constrain the grammar in the same way.

The case information can be used in a VP rule as below:

```
vp --> v, np(acc).
```

Compare this rule to the corresponding rule in hw5part1case.pl. They do the same thing. Now look at the grammar in hw5part2case.pl, which is similar to hw5part1case.pl, but implements case using features instead.

Examine hw5part2case.pl carefully, comparing it to hw5part1case.pl. Then examine hw5part2template.pl, which includes extra features, which may be useful for implementing subject-verb agreement and verb subcategorization.

Your task: starting from hw5part2template.pl, implement case, subject-verb agreement and verb subcategorization using features.

When you are done, the query

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

should give you only grammatical sentences as values of S.

Again, your grammar should be principled, accurate and simple. It is trivial to implement a grammar that produces the correct sentences somehow, but this is not sufficient. *Your grammar should implement subject-verb agreement and verb subcategorization using features in a way that does not result in an unnecessary increase in the number of grammar rules, and reflects how transitivity and subject-verb agreement work in English.*

What to turn it: your grammar, in a file called hw5part2.pl.