Language and Computation – Grammar Engineering

1. Implementation

A context-free grammar (CFG) consists of a set of recursive rules (productions), that can be used to generate patterns of strings/language [1]. It has two main purposes, to generate sentences, and to assign a structure to a given sentence.

Table 1 explains the terminal and nonterminal symbols components of an CFG.

Components of CFG	
Component	**Description**
Terminal symbols	Symbols that correspond to words in the language. They are always on the right-hand side of the production rules.
Nonterminal symbols (variables)	The symbols that express clusters or generalizations of the terminals. The non-terminals that are associated with words are their part-of-speech.

Table 1: Components of a CFG.

Each of the rules in a CFG expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols. For example, the production rule *NP -> Det Noun* expresses that a noun phrase (NP) can be composed of a determiner (Det) followed by a noun (Noun). Multiple rules can be **combined** to form the entire set of grammar rules.

The concept of derivation defines a language. One string yields another if it can be rebuilt as the second one using a set of rules. A sequence of rule expansions can therefore be used to generate a set of strings, and this is commonly represented using a parse tree as shown in Figure 1.

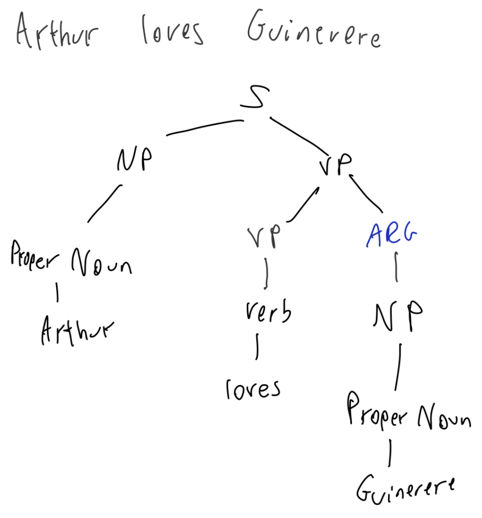


Figure 1: Tree representation of a derivation of the sentence "Arthur loves Guinevere".

1.1. The need for Agreement and Subcategorization

In English, nouns are frequently indicated as singular or plural. The demonstrative pronoun must match the noun's form (both must be singular or plural forms). In the examples below, the demonstrative “this” (singular) changes to “these” in plural form:

This parrot
These parrots

Furthermore, the syntactic aspects of the subject noun phrase co-vary with the morphological elements of the verb. When we look at verb agreement in English, we can see that present tense verbs frequently have two inflected forms: one for the third-person singular and another for every other person and number combination [3]. The agreement paradigm for English regular verbs is shown in an example in Table 2.

Agreement Paradigm - English		
Person	**Singular**	**Plural**
First	I love	We love
Second	You love	You love
Third	He/she/it loves	They love

Table 2: Agreement paradigm for verb love.

The third person-singular (3sg) form usually has a final -s where the non-3sg forms do not. One option to broaden our grammar to deal with this agreement phenomenon is to include two sets of rules, one for 3sg subjects and one for non-3sg subjects. The primary issue with this is that every rule that relates to a noun or a verb must have a "singular" and a "plural" version, thereby significantly prolonging the size of our language. Duplicate rules would also be required to capture the fact that head nouns and their determiners must agree in number, as well as rules to capture gender agreement in other languages (e.g., Italian, French, German), further expanding our grammar rules [2].

An alternative solution that was employed is to introduce **variables** over values in the production rules and categories.

S -> NP[NUM=?n] VP[NUM=?n]

In the example above we are using *?n* as a variable over values of NUM. The variable can be initialised as either sg or pl when initialising terminals. This enforces agreement between the Noun phrase and the Verb phrase, simply and elegantly. The inclusion of this stops our grammar from generating ungrammatical in terms of number agreement sentences.

Another concept is that of subcategorization of the verb – which specifies the different combinations of arguments that different verbs can take. This process is idiosyncratic – having to be found for each verb individually. For example, the verb likes can have a noun phrase (NP) as an argument (among other categories):

Maria likes [NP the food]

Subcategorization was introduced in the grammar by the inclusion of the SUBCAT parameter in all the verb related labels (VP, MD, V) which enables to specify of the syntactic category that verbs can take. This further eliminates the overgeneration of ungrammatical sentences.

For this practical, a grammar for a small subset of English was engineered. At first, the grammar was a pure context-free grammar but at a later stage number agreement features and subcategorization were also included creating a unification grammar.

1.2. Categories

When creating my grammar, I chose to use category labels that are widely accepted in the linguistics society and used in other popular grammars (Table 2). Please keep in mind that I oversimplified some steps and more category labels would be required in a larger and more complex grammar.

Labels Used – Different Levels		
Bracket Labels Level	**Category / Tag**	**Purpose and Description**
Clause Level	S	Simple declarative clause – used to start sentence.
	SBAR	Expands to a subordinating conjunction followed by another declarative sentence (NP VP) structure, allowing the creation of complex **conditional** sentences.
Phrase Level	WhNP	Start a sentence by a *wh* word.
	NP	Noun phrases (NP) are frequently made up of a head, which is the principal noun, as well as multiple modifiers that might appear before or after the head noun.
	Nominal	Nominal follows the determiner and contains any pre and post- head noun modifiers.
	VP	The verb phrase (VP) is made up of the verb and several other elements.
	PP	Prepositional phrases. Consitisting of a preposition and its object.
	ARG	An Argument of a clause.
Word Level	IN	Preposition (e.g., in, at, on) or subordinating conjunction (e.g., when, although, so).
	CC	Coordinating conjunction. Joining parts of the sentence together.
	Wh-adv	Represents wh-words that are an adverb. In our lexicon, only when.
	Det	A Determiner (e.g., the, a, an)
	V[+AUX]	Auxiliary verbs (e.g., do, does). Used to create *Wh-non-subject* question
	Noun	Noun, singular or plural (e.g., man, woman, teacher, home, office, car, table).
	MD	Modal verbs (e.g., may, should, must)
	V	Verb (e.g., run, fight, ride, work).
	VBG	Verb, gerund or present participle (e.g. drinking).

Table 3: Category/ tag labels used in the developed grammar.

1.3. Approach

I approached this practical by breaking down the whole process into smaller parts. I started by finding the constituents of each sentence and drawing each sentence's parse tree (See Appendix. I then wrote all the rules for each sentence on temporary cfcg files (one for each sentence). This process was followed separately for all three parts (pure CFG, subcategorisation, and number agreement) of this assignment.

To write the rules for each sentence, I found that categorising each positive sentence to its appropriate sentence-level construction (e.g. declarative, imperative, etc.) was a very useful first step. Table 3 shows the sentence level construction and sample rule for each sentence.

Sentence Level Construction - Positive Sentences		
Sentence	**Structure**	**Sample Rule**
Arthur loves Guinevere	Declarative	S -> NP VP
Guinevere sighs	Declarative	S -> NP VP
Guinevere and Arthur sigh	Declarative	S -> NP VP
horses drink water	Declarative	S -> NP VP
Arthur rides the horse	Declarative	S -> NP VP
Guinevere rides on Mondays	Declarative	S -> NP VP
Arthur rides	Declarative	S -> NP VP
Guinevere rides the old white horse near the castle on Mondays	Declarative	S -> NP VP
when Arthur drinks beer Guinevere sighs	Wh-phrase	S -> SBAR NP VP
Guinevere thinks Arthur drinks water	Declarative	S -> NP VP
Arthur rides near the castle and drinks from the chalice	Declarative	S -> NP VP
Arthur rides and drinks beer and water	Declarative	S -> NP VP
when do Arthur and Guinevere drink beer	Wh-subject-question	S -> WhNP Aux NP VP
when does Guinevere ride	Wh-subject-question	S -> WhNP Aux NP VP

Table 4: Sentence level construction and sample rule for each of the positive sentences.

Most of the basic positive examples use declarative sentence-level composition, with a subject noun phrase followed by a verb phrase. The final two phrases have a Wh-subject-question structure where their structure is identical to that of declarative sentences except that the first noun phrase contains some wh-word. Knowing the structure and subsequently, the sample rules for each sentence helped me greatly in constructing each sentence parse tree and gave me a significantly better understanding of how sentences are structured in English (as a non-native speaker).

After writing the grammar rules and testing each sentence individually, I then **combined** and tested all the rules and terminals for all the sentences - keeping only the distinct elements from each temporary sentence file. This process was followed for all three steps of this practical, as it allowed me to incrementally build my grammar and test it thoroughly at each step of the way.

1.3.1 Interesting choices

An interesting design choice I made can be seen in the rule used to capture conditional sentences such as "when Arthur drinks beer Guinevere sighs". The rule used S -> SBAR NP VP, suggests that a clause starting with a subordinating conjunction, must be followed by a sequence of a noun phrase followed by a verb phrase, as shown below:

$_{SBAR}$[$_{IN}$ when $_{NP}$[Arthur] $_{VP}$ [drinks beer]] $_{NP}$[Guinevere] $_{VP}$[sighs]

There are two parts to this sentence, the condition – "when Arthur drinks beer", and the result – "Guinevere sighs". They could exist as two separate sentences, however, by putting subordinating conjunction (when) at the beginning we create a conditional sentence. The SBAR rule expands to subordinating conjunction followed by an NP VP, allowing the creation of even more complex conditional sentences such as "when Guinevere rides the horse and drinks water from the chalice Arthur drinks beer near the castle". To handle the case when the condition is at the end of the sentence e.g., "Guinevere loves Arthur when the horses drink water", I wrote a new rule S -> NP VP SBAR that simply reverses the order, having the SBAR at the end.

Another intriguing design decision I made was to convert an argument (ARG) into an S rule. The S rules are meant to account for whole sentences that act as basic units of speech on their own – comparable to phrases that form a complete thought. They can therefore also exist on the right side of grammatical rules in phrases where their clauses can act as independent phrases [2]. For example, this rule (ARG[CAT=s] -> S) was applied in the argument of the word "thinks" in the sentence "Guinevere thinks Arthur drinks water". This is because the ARG of thinks - "Arthur drinks water", can act as a whole sentence on its own as it is a complete thought (Figure 2).

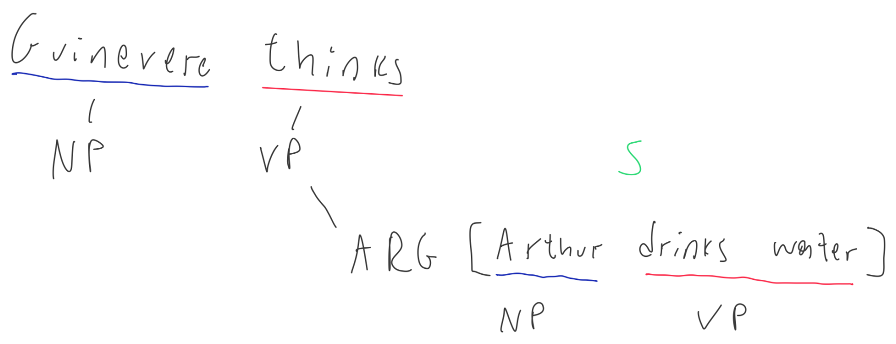


Figure 2: Whole sentence as argument.

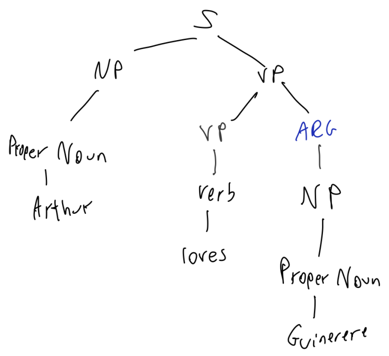
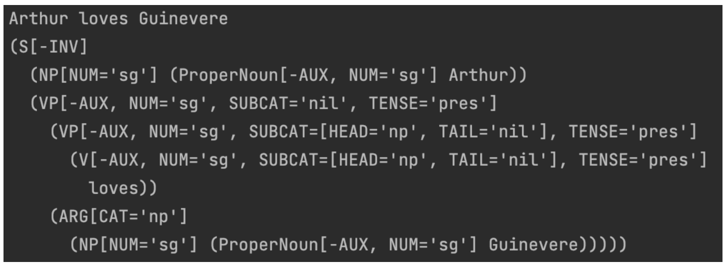
Initially, when I was writing the terminals to test my grammar, I was assigning a singular or plural NUM agreement variable to all the appropriate productions (e.g., nouns). However, after rereading Chapter 9 of the NLTK book [3], I decided to include an explicit NUM specification only if there was another value elsewhere in the production that **constrains** the value, such as having "this" (Singular), and "these" (Plural). Therefore, for productions such as 'water' and 'castle' where they do not have another constraining value in the production (e.g., waters), no NUM specification was added.

A goal of mine was to make my grammar dynamic, by for example, not restricting the number of adjectives in front of noun to a fixed predetermined size. To achieve that I took advantage of the **recursive** nature of production rules, creating recursive calls that allow for infinite number of adjectives in front of a noun. This is because there are no limits in how many adjectives can appear in front of a noun.

2. Testing – Positive Sentences

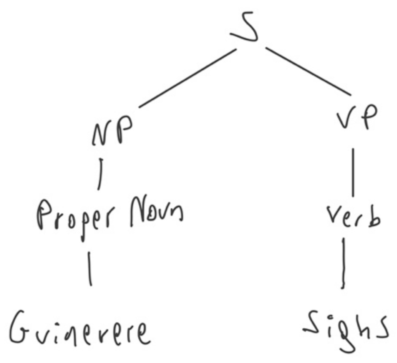
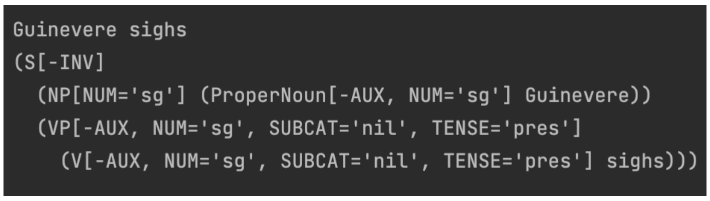
This section tests the parsing of the basic positive phrases of this assignment by comparing them with the hand drawn syntax trees that were drawn initially before writing the production rules in the grammar.

a) Arthur loves Guinevere

 S NP VP Proper Noun Arthur VP verb loves ARG NP Proper Noun Guinevere	
Correct:	**YES**

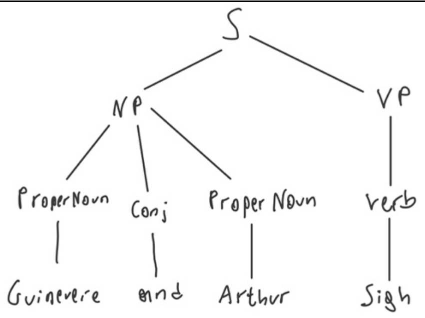
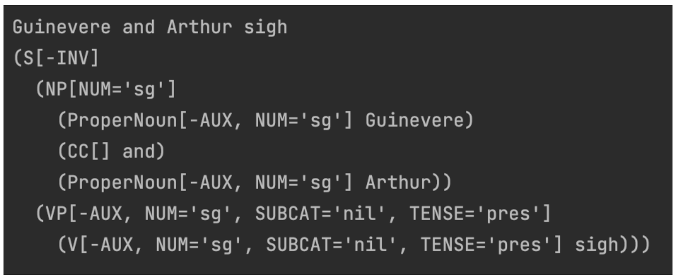
```
Arthur loves Guinevere
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
        loves))
    (ARG[CAT='np']
      (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere)))))
```

b) Guinevere sighs

	
Correct:	**YES**

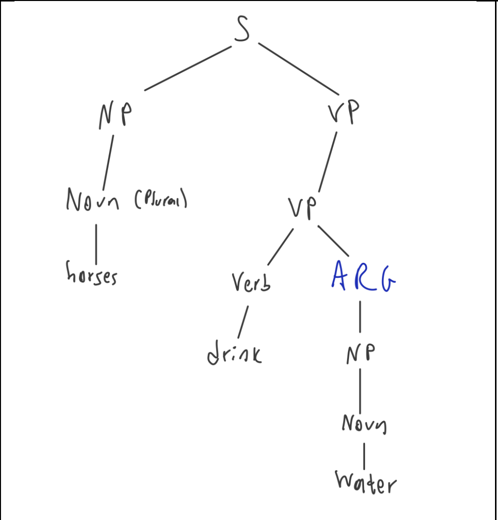
```
Guinevere sighs
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] sighs)))
```

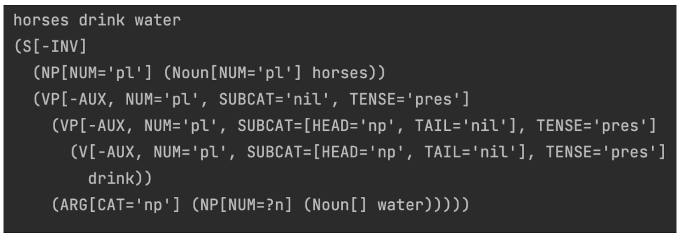
c) Guinevere and Arthur sigh

 S NP VP ProperNoun Conj Proper Noun verb Guinevere and Arthur Sigh	
Correct:	**YES**

```
Guinevere and Arthur sigh
(S[-INV]
  (NP[NUM='sg']
    (ProperNoun[-AUX, NUM='sg'] Guinevere)
    (CC[] and)
    (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] sigh)))
```

d) horses drink water

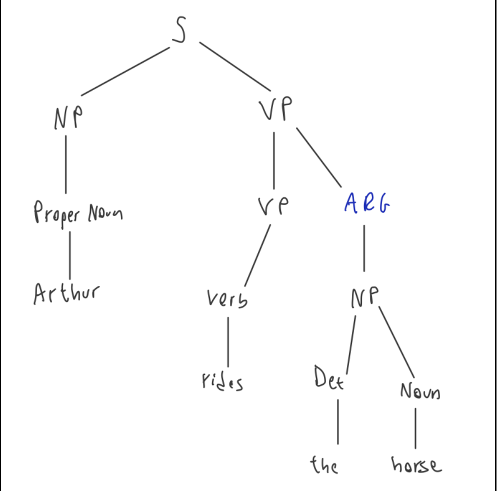


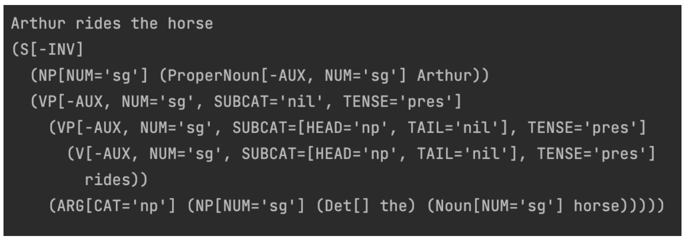


```
horses drink water
(S[-INV]
  (NP[NUM='pl'] (Noun[NUM='pl'] horses))
  (VP[-AUX, NUM='pl', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='pl', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='pl', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
        drink))
    (ARG[CAT='np'] (NP[NUM=?n] (Noun[] water)))))
```

Correct:	YES

e) Arthur rides the horse

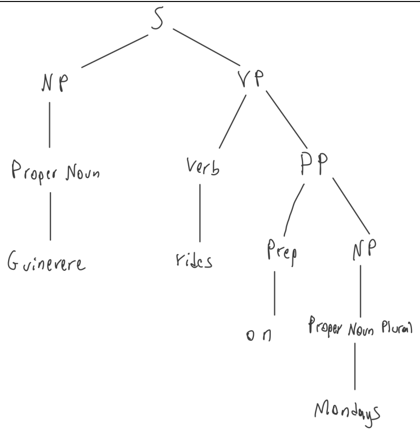


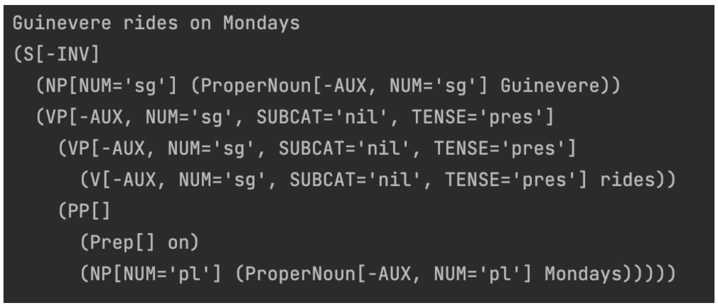


```
Arthur rides the horse
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
        rides))
    (ARG[CAT='np'] (NP[NUM='sg'] (Det[] the) (Noun[NUM='sg'] horse)))))
```

Correct:	YES

f) Guinevere rides on Mondays

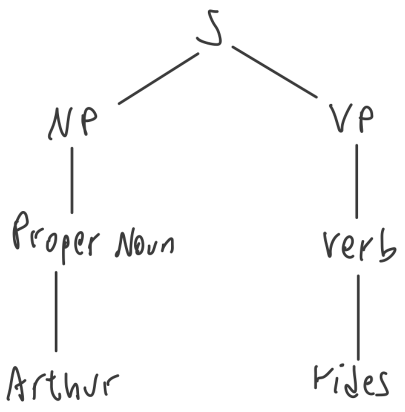


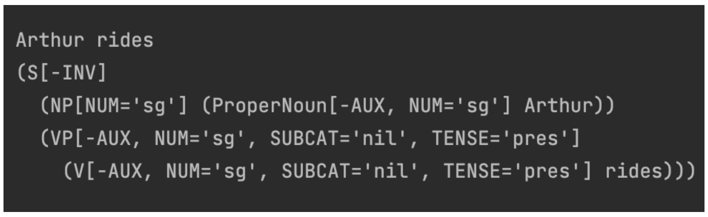


```
Guinevere rides on Mondays
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] rides))
    (PP[]
      (Prep[] on)
      (NP[NUM='pl'] (ProperNoun[-AUX, NUM='pl'] Mondays)))))
```

Correct: YES

f) Arthur rides

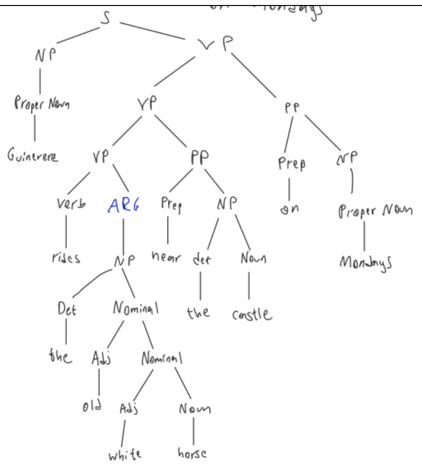


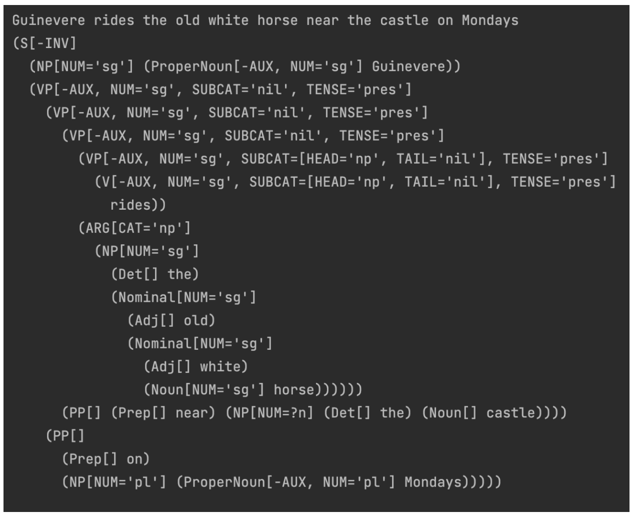


```
Arthur rides
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] rides)))
```

Correct: YES

g) Guinevere rides the old white horse near the castle on Mondays

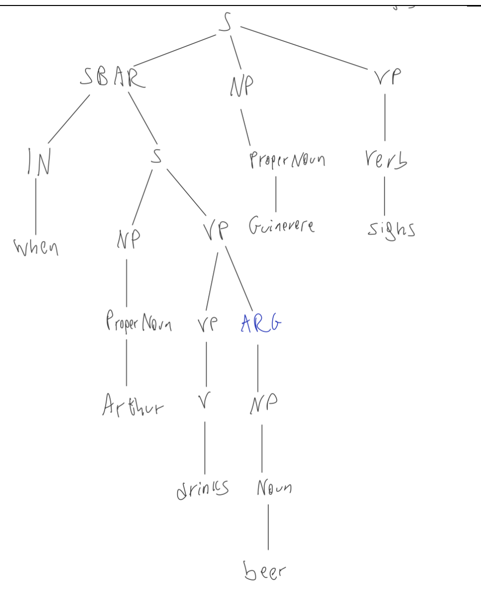


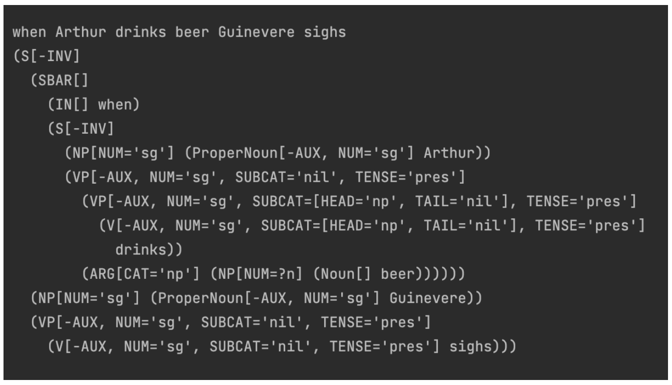


```
Guinevere rides the old white horse near the castle on Mondays
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
      (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
        (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
          (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
            rides))
        (ARG[CAT='np']
          (NP[NUM='sg']
            (Det[] the)
            (Nominal[NUM='sg']
              (Adj[] old)
              (Nominal[NUM='sg']
                (Adj[] white)
                (Noun[NUM='sg'] horse))))))
      (PP[] (Prep[] near) (NP[NUM=?n] (Det[] the) (Noun[] castle))))
    (PP[]
      (Prep[] on)
      (NP[NUM='pl'] (ProperNoun[-AUX, NUM='pl'] Mondays)))))
```

Correct: YES

h) when Arthur drinks beer Guinevere sighs

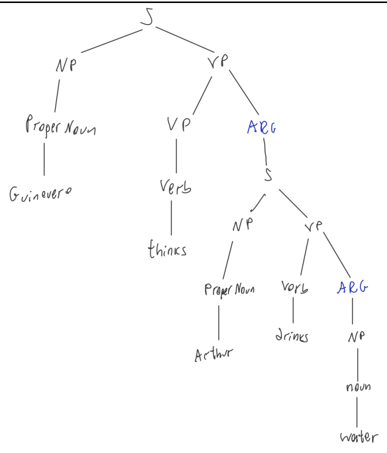


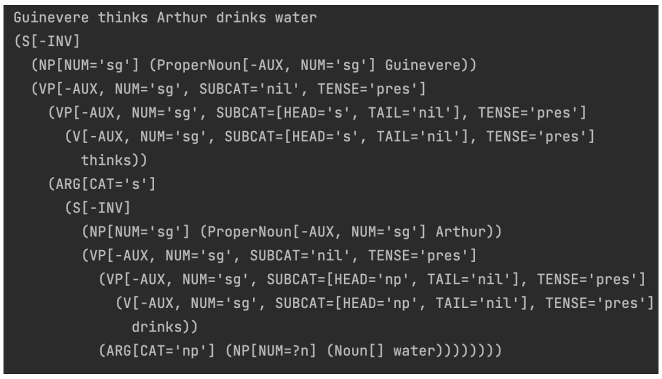


```
when Arthur drinks beer Guinevere sighs
(S[-INV]
  (SBAR[]
    (IN[] when)
    (S[-INV]
      (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
      (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
        (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
          (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
            drinks))
        (ARG[CAT='np'] (NP[NUM=?n] (Noun[] beer))))))
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] sighs)))
```

Correct:	YES

i) Guinevere thinks Arthur drinks water

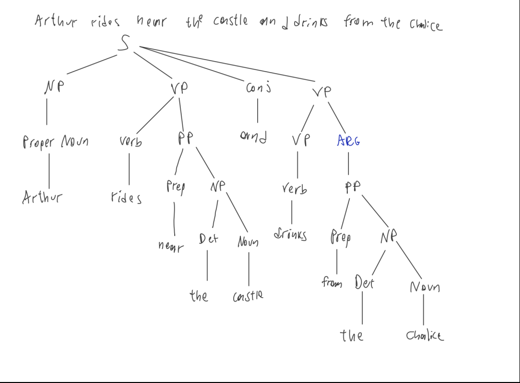


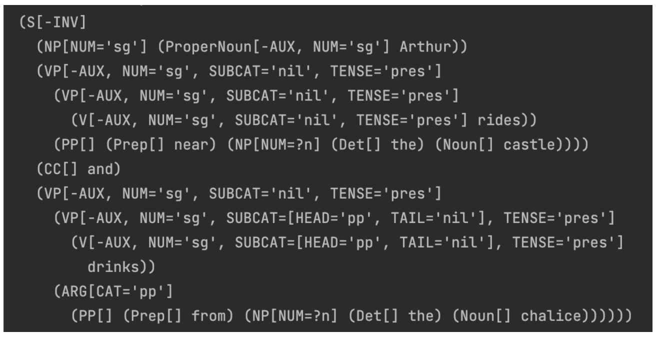


```
Guinevere thinks Arthur drinks water
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='s', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='s', TAIL='nil'], TENSE='pres']
        thinks))
    (ARG[CAT='s']
      (S[-INV]
        (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
        (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
          (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
            (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
              drinks))
          (ARG[CAT='np'] (NP[NUM=?n] (Noun[] water)))))))))
```

Correct:	YES

j) Arthur rides near the castle and drinks from the chalice

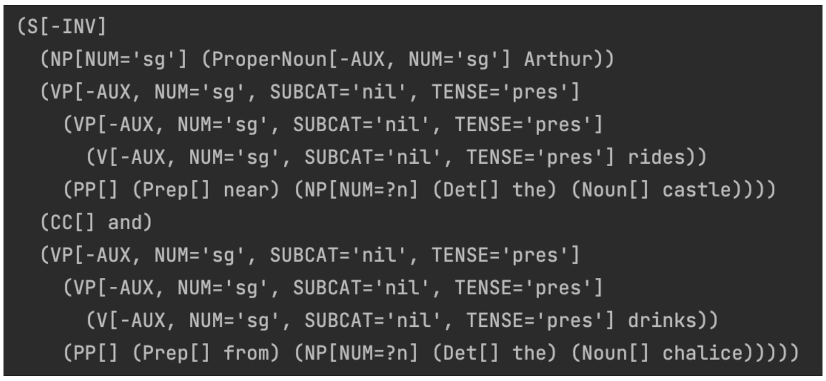




```
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] rides))
    (PP[] (Prep[] near) (NP[NUM=?n] (Det[] the) (Noun[] castle))))
  (CC[] and)
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='pp', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='pp', TAIL='nil'], TENSE='pres']
        drinks))
    (ARG[CAT='pp']
      (PP[] (Prep[] from) (NP[NUM=?n] (Det[] the) (Noun[] chalice))))))
```

Correct: **YES but overgenerated**

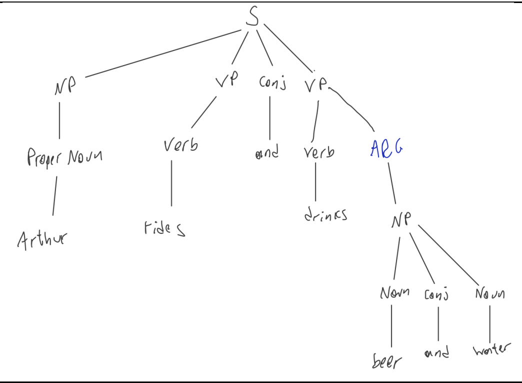
There is however another output returned by the parser:

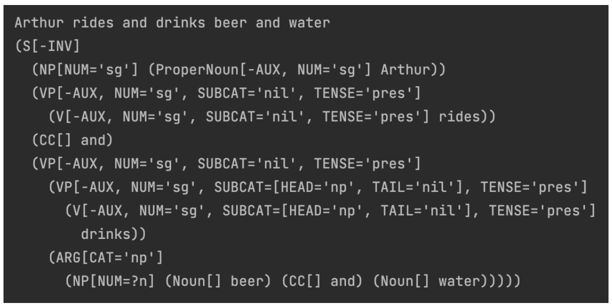


```
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] rides))
    (PP[] (Prep[] near) (NP[NUM=?n] (Det[] the) (Noun[] castle))))
  (CC[] and)
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] drinks))
    (PP[] (Prep[] from) (NP[NUM=?n] (Det[] the) (Noun[] chalice)))))
```

When attempting to include the extension sentence "what does Guinevere think Arthur drinks," the word "drinks" had to be introduced with a nil HEAD subcategory as the entire sentence's argument is an S – "Arthur drinks," and therefore when this S is expanded into NP VP, no further ARG is required. This causes the "Arthur rides near the castle and drinks from the chalice" sentence to parse two sentences one with an ARG for the "from the chalice" part and another one that uses this *nil* drinks to expand the VP into VP PP without argument. Before attempting to include the extension phrase mentioned, this output was **not** returned.

k) Arthur rides and drinks beer and water

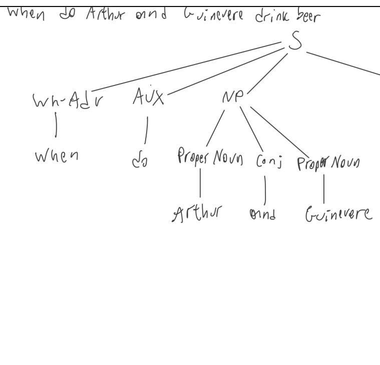
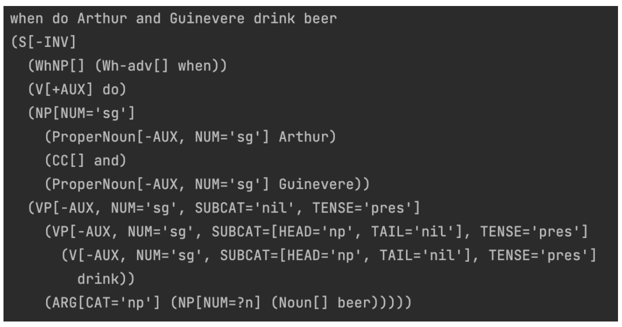




```
Arthur rides and drinks beer and water
(S[-INV]
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Arthur))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] rides))
  (CC[] and)
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
        drinks))
    (ARG[CAT='np']
      (NP[NUM=?n] (Noun[] beer) (CC[] and) (Noun[] water)))))
```

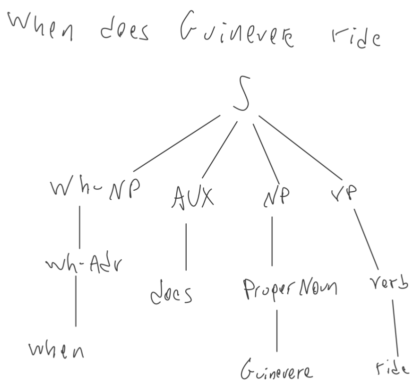
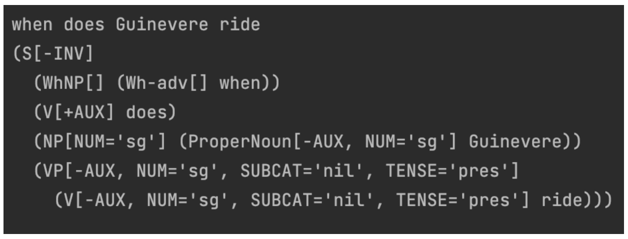
Correct: **YES**

1) when do Arthur and Guinevere drink beer

	
Correct:	**YES**

```
when do Arthur and Guinevere drink beer
(S[-INV]
  (WhNP[] (Wh-adv[] when))
  (V[+AUX] do)
  (NP[NUM='sg']
    (ProperNoun[-AUX, NUM='sg'] Arthur)
    (CC[] and)
    (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (VP[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
      (V[-AUX, NUM='sg', SUBCAT=[HEAD='np', TAIL='nil'], TENSE='pres']
        drink))
    (ARG[CAT='np'] (NP[NUM=?n] (Noun[] beer)))))
```

1) when does Guinevere ride

	
Correct:	**YES**

```
when does Guinevere ride
(S[-INV]
  (WhNP[] (Wh-adv[] when))
  (V[+AUX] does)
  (NP[NUM='sg'] (ProperNoun[-AUX, NUM='sg'] Guinevere))
  (VP[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres']
    (V[-AUX, NUM='sg', SUBCAT='nil', TENSE='pres'] ride)))
```

3. Comments and Reflections

As explained earlier I built my grammar incrementally. This enabled me to closely examine how the grammar reacted when subcategorization was added, and then how it reacted when number agreement was subsequently introduced. Subcategorisation eliminated the parsing of sentences such as "Arthur loves thinks" as it constrains the set of arguments that can be used, and a verb cannot follow another verb (except if gerund (-ing form) or of an infinitive with *to*). Furthermore, before the inclusion of number agreement handling, sentences such as "the horses drinks water" or "Guinevere love the horse" were able to be parsed by the grammar, even though the noun and verb do not agree (e.g., one horse drinks, two horses drink). After combining all the parts, the grammar successfully accepts all the positive examples provided and none of the ungrammatical negative sentences.

To further test and ensure that my grammar works as expected I did the following steps:

1. Did manual testing of the original sentences – comparing the output with the syntax tree drawn before production rules were written (See previous section)
2. Added ten additional **negative** sentences
3. Added the **positive** sentences listed in the extensions section of the specification
4. Added additional five **positive** sentences to further check specific aspects of my grammar.

Given more time, I would add more words to the grammar's lexicon and test more complex sentences, both for positive and negative cases. Based on the extra testing performed I believe that the developed grammar behaves correctly for the **implemented types** of sentences. I reflect on the fact that this is a very simple grammar that of course does not capture all the rules present in the English language but rather a small subset of them.

In the Testing section, it was found that there is only one sentence that produces more than two parses – "Arthur rides near the castle and drinks from the chalice". The grammar still captures the correct phrase (including the ARG) but it also finds another possible parsing, which was introduced when the word drinks with *nil* subcategory was added (to satisfy the phrase "what does Guinevere think Arthur drinks" in suggested extensions of the specification). Given more time, I would alter correlating production rules so that this would be eliminated, leaving only the appropriate parsing (with the ARG). Furthermore, I would have included "Inversion" (e.g., S[+INV] -> V[+AUX] NP VP) to be able to capture inverted clauses (subject and verb is switched) in a more elegant way.

Reflecting on the overall experience of this practical, I must admit that I initially found it quite challenging to get started as I am a non-native speaker of English, and I was therefore unfamiliar with many basic rules. After re-watching the lectures and reading the SLP book, I felt I had much better foundations and I found the whole process easier and very rewarding. I imagine that the task of writing the rules for all the sentences combined from the beginning would be more challenging compared to my methodology where I broke down the task into much smaller subtasks before combining them. Drawing the syntax trees by hand before starting to write the production rules was a vital step as it significantly enhanced my understanding of grammar and syntax trees. In general, it was a very fun and rewarding practical which allowed me to become a better computational linguist (if I dare to say so) and apply in practice what we have been learning in lectures for the past weeks.

4. Extensions

1. Additional Negative Sentences

As explained in the section above, I added ten additional negative sentences from the given lexicon to further test the developed grammar. Table 5 shows all the additional negative sentences tested and the reason that they are negative.

Additional Negative Sentences	
Sentence	**Reason for negative**
Guinevere near horse	Should have "is" after Guinevere and determiner "the" before "horse".
Arthur rides white	Missing noun (e.g. horse) after adjective "white".
horses loves old Arthur	Plural horses → verb must becomes love.
the white horses sigh when Mondays	The proper noun Mondays is out of context there. Was expecting a condition.
Arthur thinks beer	Expecting a preposition (e.g., of) between "thinks" and "beer".
Guinevere love the horse	Third person singular of "love" should become "loves".
white horse beer	Was expecting a verb after "horse" and before "beer".
the Arthur loves Guinevere	The determiner "the" is syntactically incorrect in this sentence.
Guinevere rides near old castle	Was expecting a determiner after "near" – e.g., "the".
does Arthur thinks	Should have been "think" – "does" already indicates that its 3rd person singular.

Table 5: Additional negative sentences.

As expected, **none** of the phrases was parsed from the grammar.

2. Additional Positive Sentences

The Table below lists the additional positive sentences included (suggested in the specification) alongside a comment about their structure and if they were parsed succcesfully.

Additional Positive Sentences – Suggested in Specification		
Sentence	**Comment**	**Parsed Correctly**
horses love drinking water	Declarative sentence (NP VP). Its argument is a Verb Phrase – drinking water. The word "drinking" is the only Gerund verb (VBG) in our simplistic lexicon.	YES
Arthur may have loved Guinevere	Declarative sentence (NP VP). The argument is a Verb Phrase – "loved Guinevere". The word "may" is a modal verb.	YES
Arthur gives Guinevere the horse	Declarative sentence (NP VP). The argument is a Noun Phrase – "Guinevere the horse".	YES
what does Arthur give Guinevere	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this sentence is **not** the wh word (what) but instead the proper noun "Arthur". The argument is "Guinevere".	YES

whom does Guinevere give the horse	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this sentence is **not** the wh word (whom) but instead the proper noun "Guinevere". The argument is the noun phrase "the Horse".	YES
whom do Guinevere and Arthur give the horse	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this question is "Guinevere and Arthur". The argument is the noun phrase "the Horse".	YES
what does Guinevere drink	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this question is "Guinevere". There is no argument in this sentence.	YES
whom does Guinevere think Arthur gives the horse	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this question is "Guinevere". There are two arguments in this sentence: 1) the whole second part of the sentence "Arthur gives the horse" 2) the part "the horse" being an argument for gives.	YES
what does Guinevere think Arthur drinks	Wh-**non**-subject-question sentence (WhNP AUX NP VP). The subject in this question is "Guinevere". The argument is the sentence "Arthur drinks".	YES

Table 6:Additional positive sentences suggested in the specification.

At the final stages of this practical I also included another argument that specifies the tense of the sentence (present or past). The past sentences tried include "Arthur may have loved Guinevere" from the suggested extensions and "the horse drank water" and "Arthur thought of beer" from below.

The five following additional phrases (Table 7) were added in the positive sentences.

Additional Positive Sentences – My own		
Sentence	**Reason for inclusion**	**Parsed Correctly**
Guinevere loves Arthur and rides the horse	Further check sentences that entail conjunction.	YES
when Guinevere rides the horse and drinks water from the chalice Arthur drinks beer near the castle	Check a longer sentence than the ones provided.	YES
the white horses sigh when Arthur drinks water	Further check conditional sentences.	YES
the horse drank water	Check that parsing for past sentences works as expected	YES
Arthur thought of beer		YES

Table 7: My own additional positive sentences.

The grammar successfully parses all these sentences.

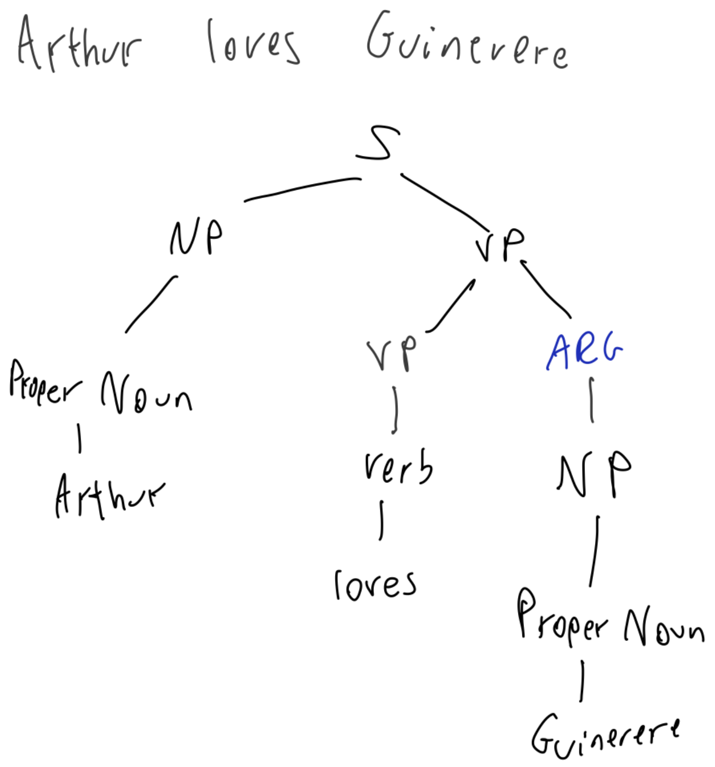
References:

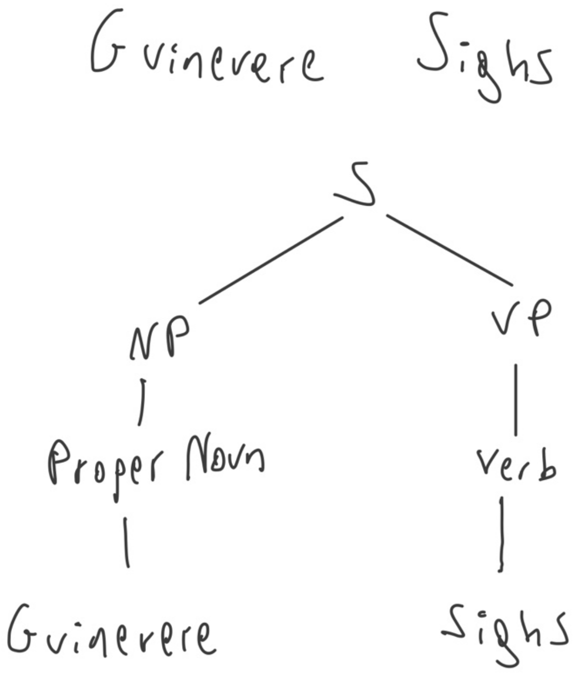
[1] Brilliant.org. n.d. *Context Free Grammars | Brilliant Math & Science Wiki*. [online] Available at: https://brilliant.org/wiki/context-free-grammars/

[2] Jurafsky, D. and Martin, J., 2009. *Speech and language processing*. Upper Saddle River, N.J.: Pearson Prentice Hall.

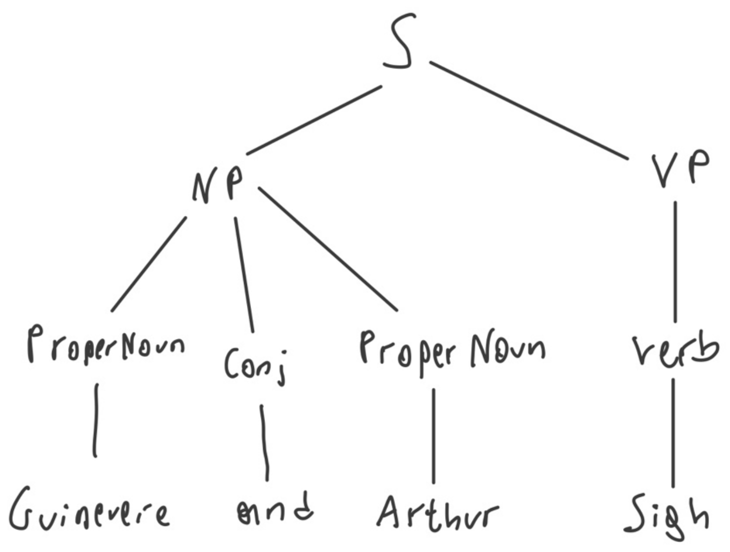
[3] Bird, S., Klein, E. and Loper, E., n.d. *Natural Language Processing with Python - Natural Language Processing with Python*. [online] Nltk.org. Available at: https://www.nltk.org/book/

Appendix – ALL HAND DRAWN SYNTAX TREES

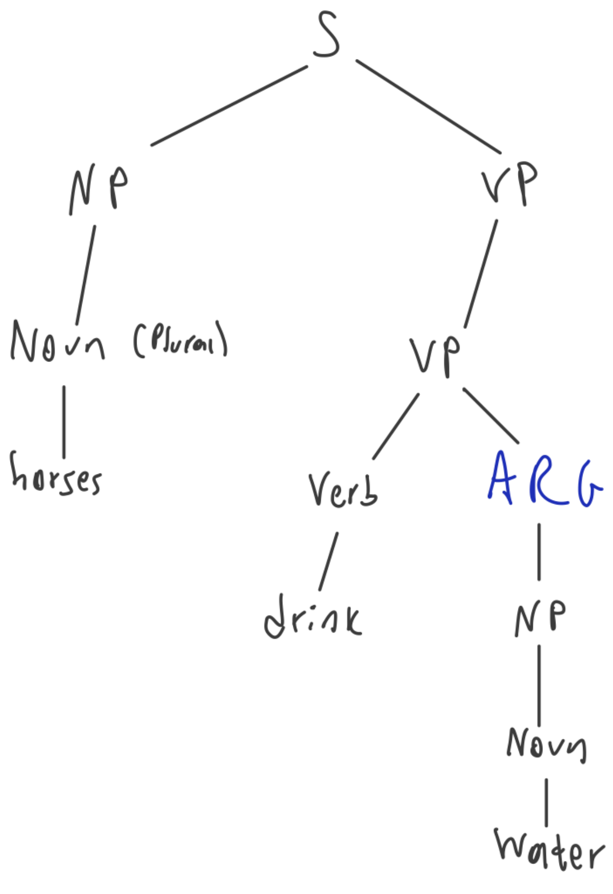


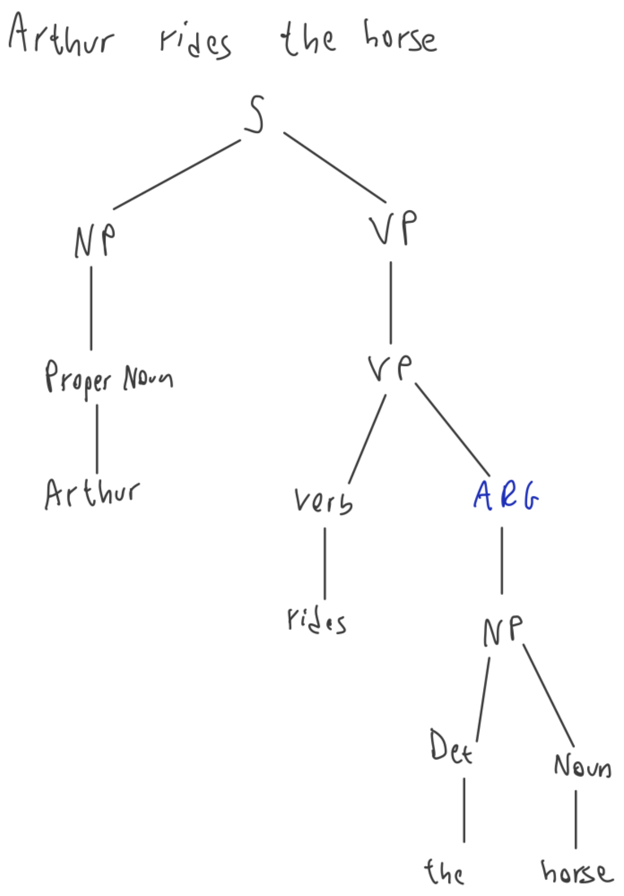


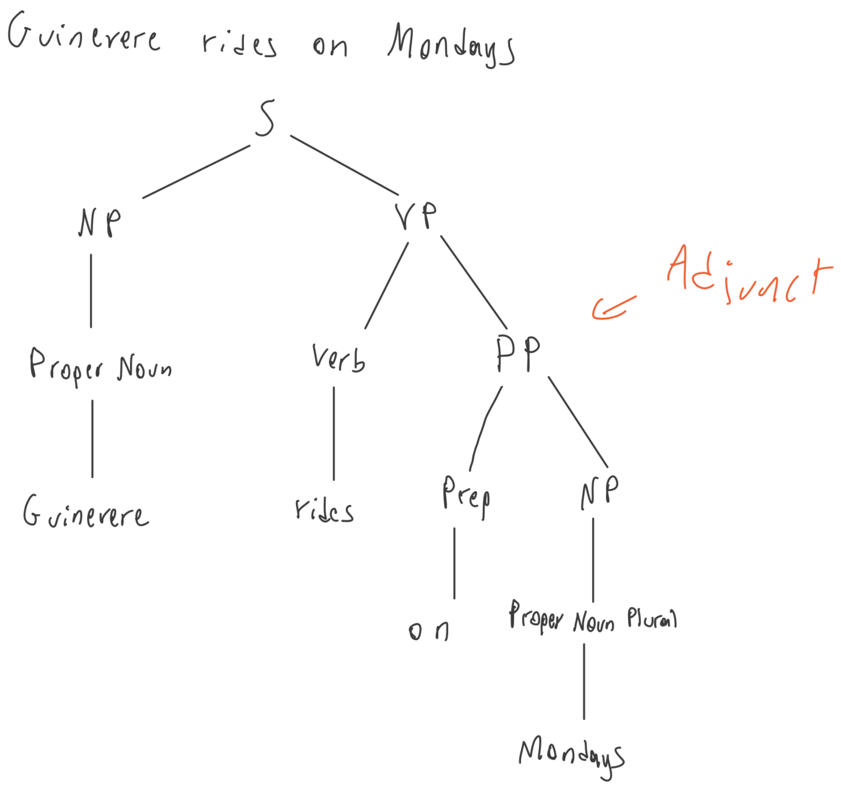
Guinevere and Arthur Sigh



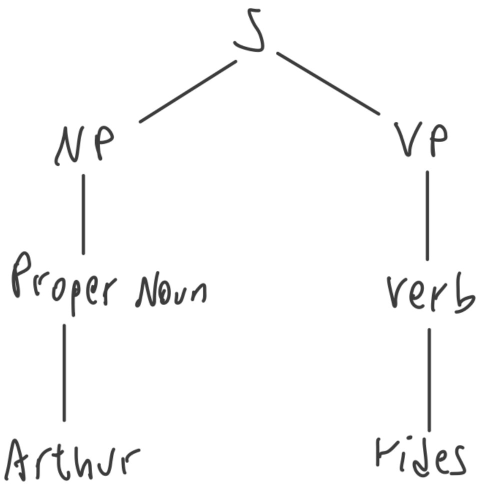
horses drink water



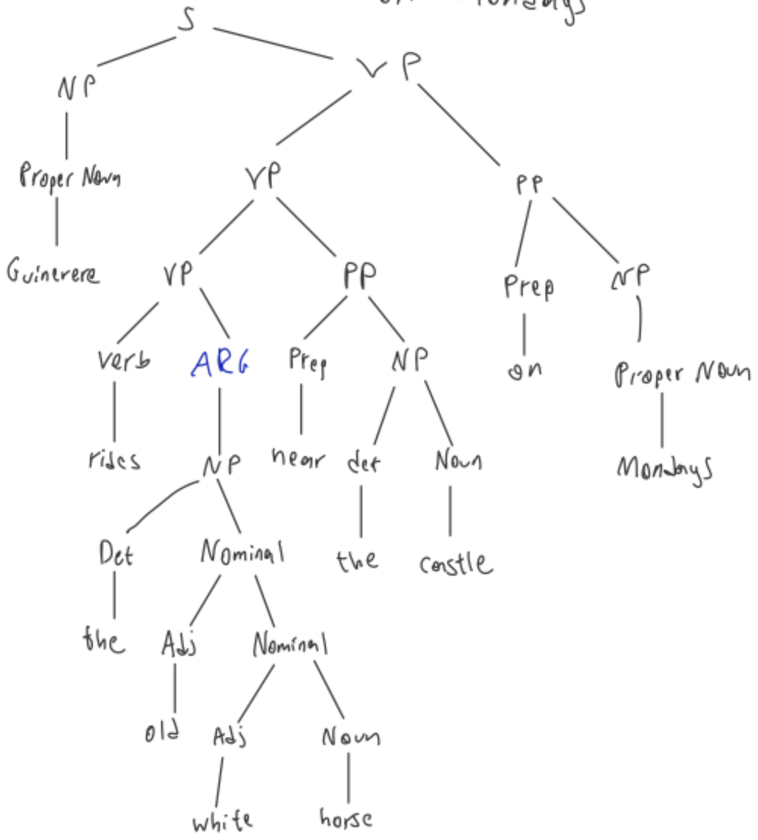

Arthur rides the horse
S
NP
VP
Proper Noun
Arthur
VP
Verb
ARG
rides
NP
Det
Noun
the
horse

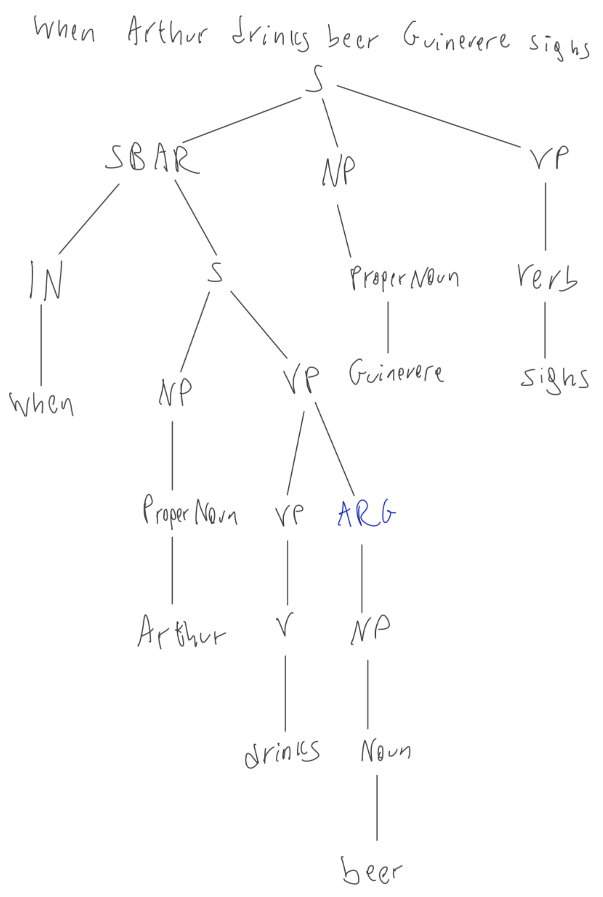

Guinevere rides on Mondays
S
NP
VP
Proper Noun
Guinevere
Verb
rides
PP
Adjunct
Prep
NP
on
Proper Noun Plural
Mondays

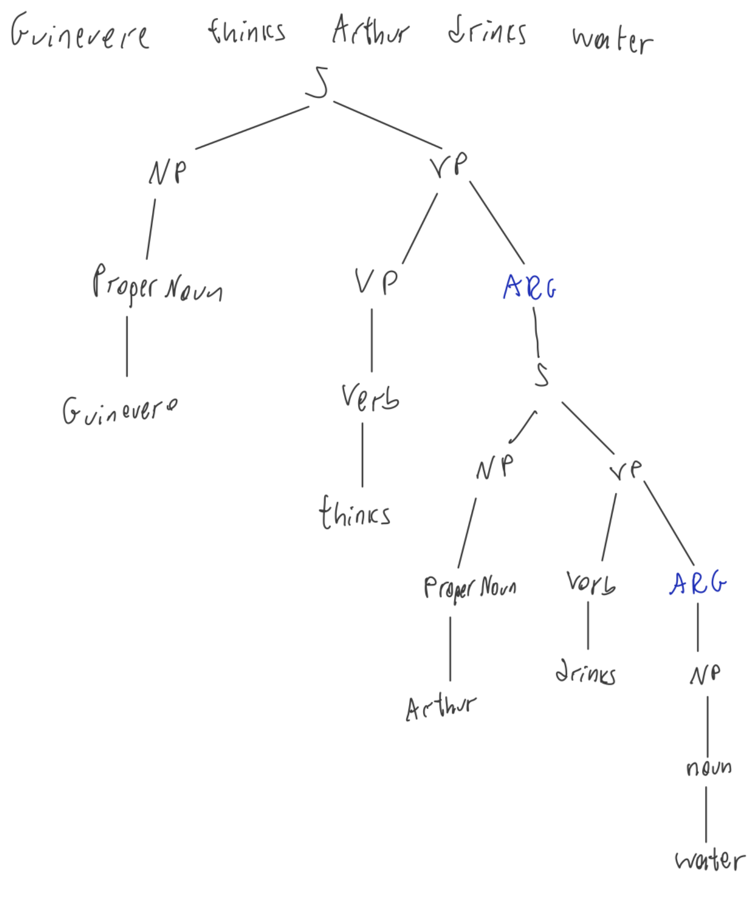
Arthur rides

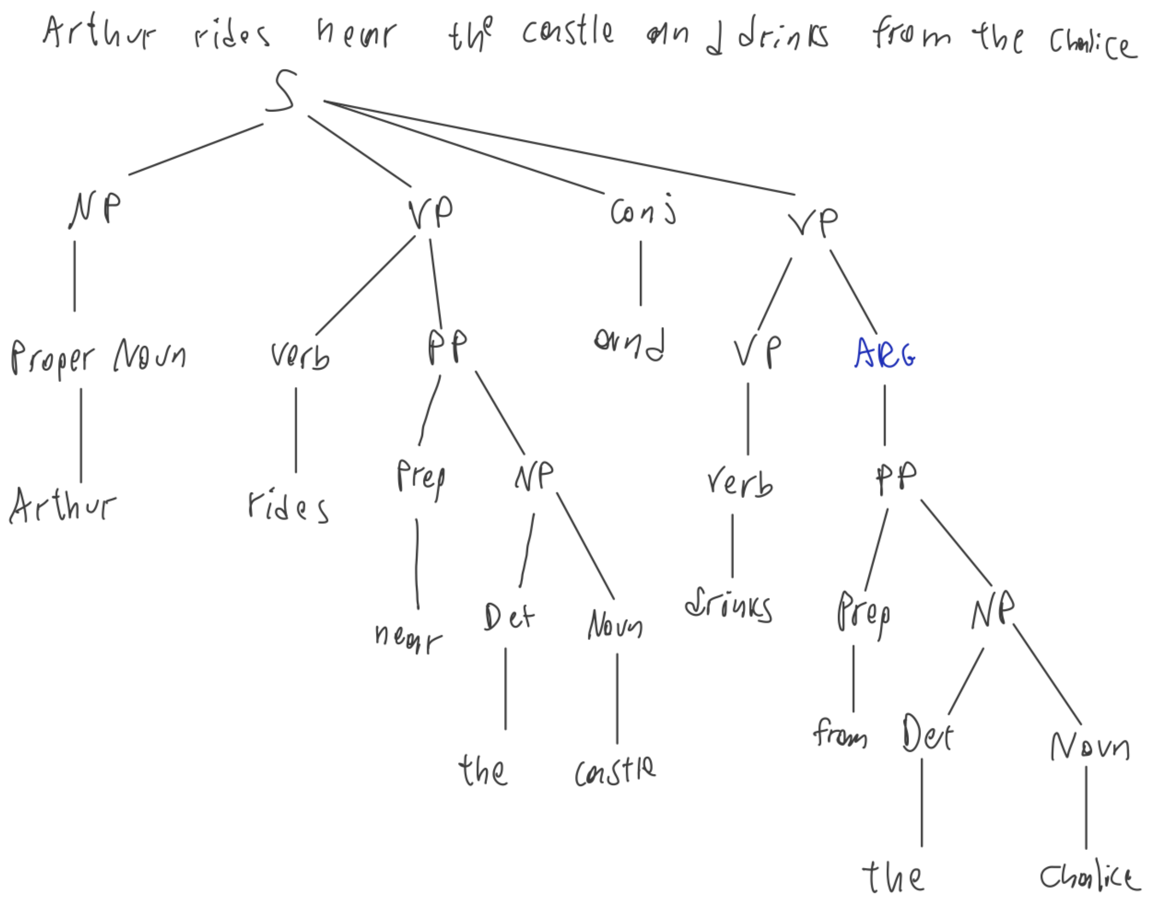


Guinevere rides the old white horse near the castle on Mondays

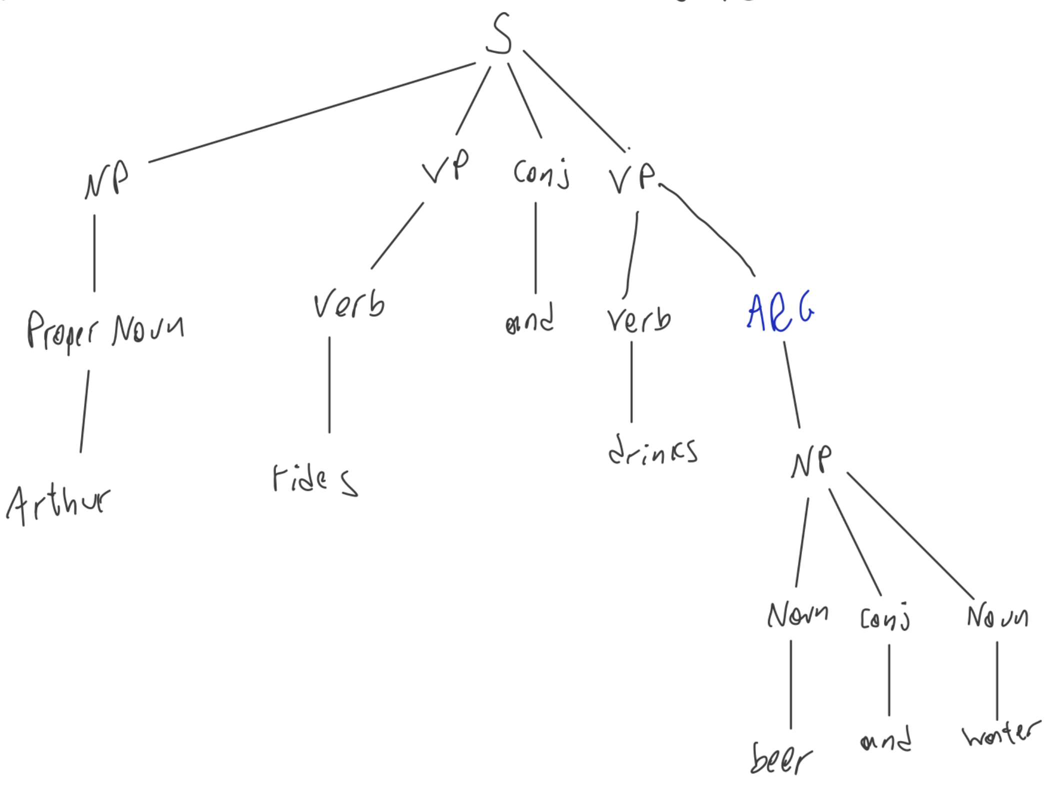



When Arthur drinks beer Guinevere sighs
S
SBAR
NP
VP
IN
S
ProperNoun
Verb
when
NP
VP
Guinevere
sighs
Proper Noun
VP
ARG
Arthur
V
NP
drinks
Noun
beer

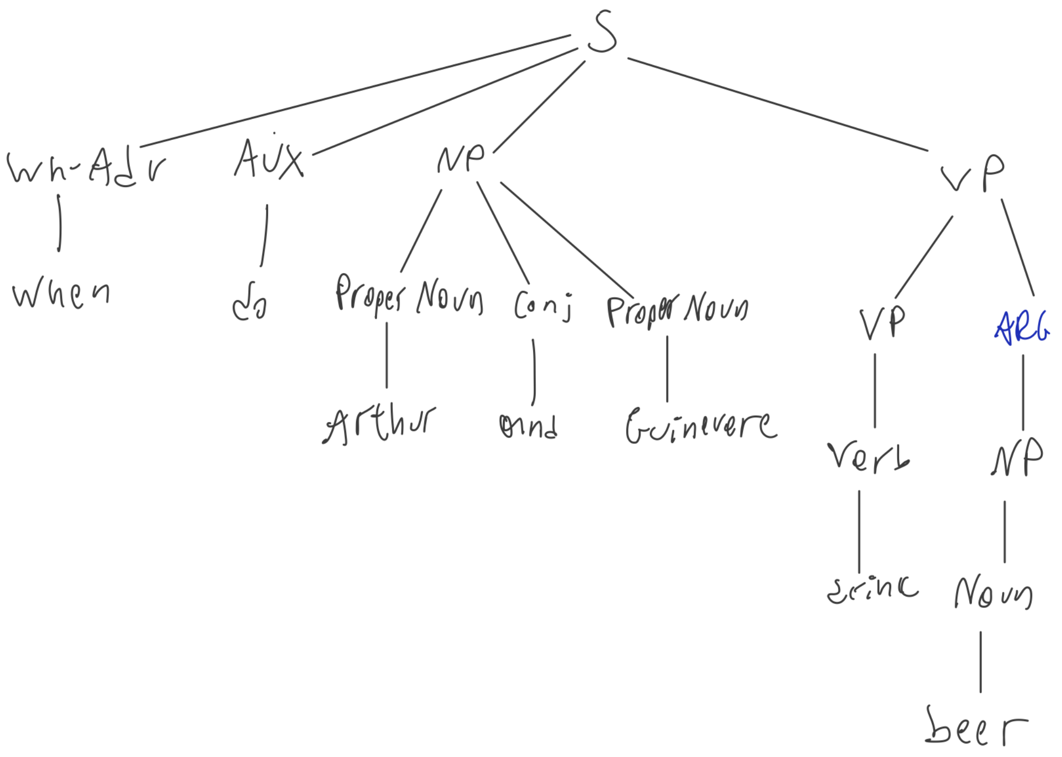

Guinevere thinks Arthur drinks water
S
NP
VP
Proper Noun
VP
ARG
Guinevere
Verb
S
thinks
NP
VP
Proper Noun
Verb
ARG
Arthur
drinks
NP
noun
water


Arthur rides near the castle and drinks from the chalice
S
NP
VP
Conj
VP
Proper Noun
Verb
PP
and
VP
ARG
Arthur
rides
Prep
NP
Verb
PP
near
Det
Noun
drinks
Prep
NP
the
castle
from
Det
Noun
the
chalice

Arthur rides and drinks beer and water



when do Arthur and Guinevere drink beer


S
Wh-Adv
AUX
NP
VP
when
do
Proper Noun
Conj
Proper Noun
VP
ARG
Arthur
and
Guinevere
Verb
NP
drink
Noun
beer