HUGH HAN, XIAOCHEN LI, SHIJIE WU

600.465 Natural Language Processing **Fall 2016**
Homework #1 **Due:** 21 September 2016, 2pm

1 Initial Implementation

The following output was produced using `./randsent grammar 10`.

```
is it true that every sandwich under every floor kissed the sandwich ?
is it true that the pickle under the pickle on a delicious pickled president pickled the
    president under the pickle ?
the president on the president in every fine floor under the sandwich in every president
    under a chief of staff ate the floor under a chief of staff on the pickle on a floor in
    every chief of staff with the perplexed president on the president .
a chief of staff pickled the sandwich in a floor on the sandwich with the president with the
    fine floor in every sandwich in a floor on a pickle on every sandwich with every sandwich
    under the pickle under a sandwich on the president under a pickled sandwich in the
    delicious floor on the pickled pickled pickle under the chief of staff in every president
    on the perplexed pickle on the pickle in every president on every floor on a sandwich in
    the chief of staff in every pickle in the pickle with a floor on the sandwich with every
    floor on a president with a chief of staff with a floor on every floor in every chief of
    staff in a sandwich on every floor on the president on the president under the chief of
    staff in every pickle under every chief of staff with every pickle with every pickle on
    every president on the pickle .
a pickle on the sandwich with every sandwich on the chief of staff under the chief of staff
    on a chief of staff under every pickle with every president in a pickle under every pickle
    on a chief of staff with the pickle under every pickle on a chief of staff under every
    delicious sandwich under a pickle with every president under the president in the sandwich
    in every pickle in the sandwich under the pickled chief of staff with every sandwich under
    the floor with a delicious floor with a pickle in a pickled chief of staff under every
    floor under every sandwich in every floor under every chief of staff under every chief of
    staff in every sandwich with the chief of staff with the president under the pickle under
    every pickle with every pickle in the floor in every president with the president with a
    pickled sandwich on a president under a floor in the fine floor in a chief of staff under
    a chief of staff under every floor in a perplexed fine floor with a president with the
    president under a president in the delicious delicious floor on every pickle on the pickle
    under a fine pickle with the chief of staff with every chief of staff on the sandwich in
    every sandwich on every chief of staff in the sandwich under a pickle in every delicious
    perplexed sandwich with the floor with a perplexed chief of staff with the president on
    the floor on the pickle in a pickle under every floor ate the chief of staff !
every delicious president wanted the fine pickle .
is it true that the pickle wanted the sandwich ?
a floor wanted a president !
every pickle with every chief of staff kissed a delicious fine president with a president .
a chief of staff with a floor pickled the perplexed perplexed fine sandwich .
```

(Note: the indentation, or “tabs”, seen in this LaTeX document were not present in the generation of these sentences. They are there to help distinguish each sentence from the other.)

2 Initial Discussion

(a) The rule

$$\texttt{NP} \rightarrow \texttt{NP PP}$$

is responsible for all these long sentences.

To explain why, let's first name the nonterminals that are not preterminals "true-nonterminals" because they do not turn into terminals in one step. Mostly, a preterminal (except "Noun") will eventually turn into one word in the final sentence, while a true-nonterminal will always turn into more than one word. Generally, true-nonterminals have more potential to generate long sentences.

Looking at all the rules that turn true-nonterminals into other symbols, we can find two rules turn a true-nonterminal to more than one true-nonterminals. they are

(i) `S` $\rightarrow$ `NP VP`,

(ii) `NP` $\rightarrow$ `NP PP`.

However, rule (i) is the only rule that turns `S` into other symbols—there are no other possible options `S` is given. (And with our grammar, we are *always* given `S` right after `ROOT`.)

When given `NP`, it's a bit different. We have the possibility of creating terminals (i.e. turning `NP` into `Det Noun`. However, if we use rule (ii) when `NP` is given, then `NP` becomes `NP PP`. Because `PP` can only turn into `Prep NP`, we are then left with an expression containing *two* `NP`s: `NP Prep NP`.

Rule (ii) creates something sort of like an explosion: one `NP` could give us two, and two `NP`s could give us even more. And this is why we generate so many long sentences!

(b) The intuition here is that we have six rules–all of equal probability–that turn the symbol `Noun` into other symbol(s), with only one of those rules generating an `Adj`. (That rule is `Noun` $\rightarrow$ `Adj Noun`).

Now let's do some calculations. Suppose there are n `Noun`s. Because the probability of any rule being applied is uniformly random, we can choose a rule for `Noun` uniformly randomly.

After one "step" of the rule is applied to all n `Noun`s in the sentence, we will have an expected $\frac{5n}{6}$ words and $\frac{n}{6}$ `Adj Noun`s. After the second step, we have an expected $\frac{5n}{6^2}$ `Adj Noun`s and $\frac{n}{6^2}$ `Adj Adj Noun`s.

In fact, we can generalize the expected number of consecutive `Adj`s:

$$\begin{aligned}
\mathbf{E}[\texttt{Adj}] &= \frac{n}{6^2} + \frac{n}{6^3} + \frac{n}{6^4} + \cdots \\
&= \sum_{i=2}^{\infty} \frac{n}{6^i} \\
&= \left(\sum_{i=1}^{\infty} \frac{n}{6^i} \right) - \frac{n}{6} \\
&= \frac{n}{5} - \frac{n}{6} \\
&= \frac{n}{30}
\end{aligned}$$

That is, the expected number of consecutive `Adj`s appearing next to each other is only one-thirtieth of the number of `Noun`s that appear in the sentence.

(c) This has been implemented in `Grammar.py`.

(d) To reduce the number of long sentences, we have to reduce the possibility that a nonterminal changes into another nonterminal. To increase the number of adjectives, we can simply increase the probability that a `Noun` changes into a `Adj Noun`.

In our grammar, we changed the rules:

(a) `1 NP` → `NP PP` into `0.2 NP` → `NP PP`,

(b) `1 Noun` → `Adj Noun` into `3 Noun` → `Adj Noun`.

Change (i) solves the problem of long sentences, in that it reduces the likelihood of a single nonterminal turning into multiple nonterminals.

Change (ii) solves the problem of too few adjectives, in that it increases the likelihood of adjective generation.

(e) The changes made for `grammar2` are:

```
2    ROOT  S .
#    ...
0.3  Verb  pickled
0.5  Det   every
0.5  Noun  pickle
0.5  Noun  chief of staff
1.5  Adj   fine
1.5  Prep  on
1.5  Prep  in
```

The reasoning for these changes are:

- a sentence ending with a period is more common than either a sentence ending with an exclamation point or a sentence ending with a question mark,
- the verb “pickled” is not frequently used in the English language,
- the frequencies are picked relative to the frequencies of other words of the same type.

Ten sentences generated with the new grammar are pasted below:

```
the chief of staff wanted a pickle .
the president pickled a president .
the floor wanted every president on the sandwich on the sandwich !
is it true that the president wanted every pickle ?
a delicious pickled pickle wanted the sandwich !
a chief of staff ate the sandwich .
the pickled floor under the president with the pickled floor wanted the fine fine floor !
a perplexed president under the fine pickled perplexed delicious sandwich ate the floor
    with the pickle with the floor .
a chief of staff understood a floor !
a floor kissed a perplexed perplexed sandwich .
```

(Note: the indentation, or “tabs”, seen in this LaTeX document were not present in the generation of these sentences. They are there to help distinguish each sentence from the other.)

3 Modifications

4 Trees, Pretty Prints, Brackets

```
is it true that {[Xiaochen] ate [Xiaochen]} ?
(ROOT is
      it
      true
      that
      (S (NP (Name Xiaochen))
          (VP (Verb ate)
              (NP (Name Xiaochen))))
      ?)

is it true that {[a very delicious chief of staff] understood [a perplexed floor]} ?
(ROOT is
      it
      true
      that
      (S (NP (Det a)
             (Noun (Adj (Adv very)
                        (Adj delicious))
                   (Noun chief
                         of
                         staff)))
         (VP (Verb understood)
             (NP (Det a)
                 (Noun (Adj perplexed)
                       (Noun floor)))))
      ?)

is it true that {[Xiaochen] wanted [a fine sandwich]} ?
(ROOT is
     it
     true
     that
     (S (NP (Name Xiaochen))
         (VP (Verb wanted)
             (NP (Det a)
                 (Noun (Adj fine)
                       (Noun sandwich)))))
     ?)

{it kissed [Xiaochen] that {[Sally] understood [Sally]}} .
(ROOT (S it
         (Verb kissed)
         (NP (Name Xiaochen))
         (N_clause that
                   (S (NP (Name Sally))
                      (VP (Verb understood)
                          (NP (Name Sally))))))
     .)

{[Xiaochen] flew} !
(ROOT (S (NP (Name Xiaochen))
         (VP (V_intran flew)))
     !)
```

5 Alternate Derivations

(a) The first (given) derivation is:

```
(ROOT (S (NP (NP (NP (Det every)
                     (Noun sandwich))
                 (PP (Prep with)
                     (NP (Det a)
                         (Noun pickle))))
             (PP (Prep on)
                  (NP (Det the)
                      (Noun floor))))
         (VP (Verb wanted)
              (NP (Det a)
                   (Noun president))))
      .)
```

An alternate derivation is as follows:

```
(ROOT (S (NP (NP (Det every)
                 (Noun sandwich))
             (PP (Prep with)
                 (NP (NP (Det a)
                         (Noun pickle)))
                     (PP (Prep on)
                         (NP (Det the)
                             (Noun floor)))))
         (VP (Verb wanted)
             (NP (Det a)
                 (Noun president))))
      .)
```

(b) In the first derivation, the sandwich was on the floor, but the pickle was not.

In the second derivation, the pickle was on the floor, but the sandwich was not.

These two sentences are both grammatically correct, but have different literal meanings in English. Therefore, it is necessary to note these differences.

6 Parse

(a) The parser does not always recover the original derivation that was "intended" by `randsent`. In fact, we can give an example of a sentence for which `parse` "misunderstood" and found an alternative derivation.

Our sentence was generated via the following tree:

```
(ROOT (S (N_clause that
                 (S (N_clause that
                             (S (NP (NP (NP (NP (Noun_pro Sally))
                                            (Conj and)
                                            (NP (Det a)
                                                (Noun (Adj (Adv really)
                                                           (Adj fine))
                                                      (Noun pickle))))
                                        (Conj and)
                                        (NP (NP (Det the)
                                                (Noun sandwich))
                                            (PP (Prep in)
                                                (NP (NP (NP (Noun_pro Xiaochen))
                                                        (PP (Prep in)
                                                            (NP (Noun_pro Sally))))
                                                    (PP (Prep with)
                                                        (NP (Det the)
                                                            (Noun sandwich)))))))
                                    (PP (Prep on)
                                        (NP (Det every)
                                            (Noun pickle))))
                                (VP (Verb wanted)
                                    (NP (Noun_pro Sally)))))
                    (VP (VP (Verb kissed)
                            (NP (Det a)
                                (Noun (Adj (Adv really)
                                           (Adj fine))
                                      (Noun sandwich))))
                        (Conj and)
                        (VP (Verb wanted)
                            (NP (Noun_pro Xiaochen))))))
         (VP (V_intran flew)))
      .)
```

(This problem is continued on the next page.)

However, parse found an alternate derivation via the following tree:

```
(ROOT (S (N_clause that
                (S (N_clause that
                           (S (NP (NP (NP (NP (NP (NP (Noun_pro Sally))
                                                  (Conj and)
                                                  (NP (NP (Det a)
                                                          (Noun (Adj (Adv really)
                                                                     (Adj fine))
                                                                (Noun pickle)))
                                                      (Conj and)
                                                      (NP (Det the)
                                                          (Noun sandwich))))
                                              (PP (Prep in)
                                                  (NP (Noun_pro Xiaochen))))
                                          (PP (Prep in)
                                              (NP (Noun_pro Sally))))
                                      (PP (Prep with)
                                          (NP (Det the)
                                              (Noun sandwich))))
                                  (PP (Prep on)
                                      (NP (Det every)
                                          (Noun pickle))))
                              (VP (Verb wanted)
                                  (NP (Noun_pro Sally)))))
                   (VP (VP (Verb kissed)
                           (NP (Det a)
                               (Noun (Adj (Adv really)
                                          (Adj fine))
                                     (Noun sandwich))))
                       (Conj and)
                       (VP (Verb wanted)
                           (NP (Noun_pro Xiaochen))))))
        (VP (V_intran flew)))
      .)
```

(Note that the two trees are not equivalent.)

The reason for this is that the “correct” tree of a specific sentence could be ambiguous, depending on the grammar that was used. That is, if there is more than one unique tree that leads to the generated sentence, the parser could guess a tree that is different from what was actually used by the generator.

In our example, the sentence was composed of several subtrees first split using the `NP Conj NP` rule, and then later on split using the `NP PP` rule. However, the parser guessed that the sentence was composed of subtrees that consecutively split using the `NP PP` rule, and did not use the `NP Conj NP` rule to split until much later on.

(b) There are 5 ways to analyze the noun phrase

`every sandwich with a pickle on the floor under the chief of staff`

using the original grammar `grammar`.

Let's first define the term "modify" as follows:

Assume that the rule $\texttt{NP}_1 \to \texttt{NP}_2$ `PP` exists in our grammar. Then `PP` modifies some $N \in$ `Noun` if $\texttt{NP}_2$ at some point generates N.

In this sentence:

(i) the `PP` "with a pickle" can only modify the `Noun` "sandwich",

(ii) the `PP` "on the floor" can modify the `Nouns` "sandwich" or "pickle",

(iii) the `PP` "under the chief of staff" can modify the `Nouns` "sandwich", "pickle", or "floor".

By this logic, there would seem to be $1 \cdot 2 \cdot 3 = 6$ ways to parse this sentence.

However, when "on the floor" modifies "sandwich", "under the chief of staff" cannot simultaneously modify "sandwich".

Thus, there remains 5 possible ways to parse the sentence.

(c) Using `grammar`, longer sentences that were generated generally had a greater number of different parses—ranging up to integers larger than $2^{31} - 1$ such that it caused a bit overflow. Shorter sentences had a smaller number—generally between 1 to 2 different parses. However, the number of different parses were either *really* small or *really* big, and rarely in between.

Using `grammar3`, longer sentences occur much less frequently due to the fixes that were implemented to reduce repetitive sentences. Because of this, sentences were generally shorter and thus had a smaller number of different parses. However, even when the sentences were somewhat long, the number of different parses was often less than 100.

We suspect that long sentences have a large number of different parses when `grammar` is used, due to a lesser number of rules, which leads to ambiguity. When long sentences occur via `grammar3`, however, the sentences are more likely to be defined by more specific rules, and are therefore more restricted and less prone to ambiguity.

(d) (i) Why is `p(best parse)` so small? Well, we can examine the probabilities used to generate this sentence, by first taking a look at the probabilities given from `grammar`.

$$\begin{aligned}\textbf{Pr}[\text{this sentence}] &= \textbf{Pr}[\texttt{ROOT}] \cdot \textbf{Pr}[\texttt{S .} \mid \texttt{ROOT}] \cdot \textbf{Pr}[\texttt{NP VP} \mid \texttt{S}] \cdot \textbf{Pr}[\texttt{Det Noun} \mid \texttt{NP}] \cdot \\ &\quad \textbf{Pr}[\texttt{the} \mid \texttt{Det}] \cdot \textbf{Pr}[\texttt{president} \mid \texttt{Noun}] \cdot \textbf{Pr}[\texttt{Verb NP} \mid \texttt{VP}] \cdot \\ &\quad \textbf{Pr}[\texttt{ate} \mid \texttt{Verb}] \cdot \textbf{Pr}[\texttt{Det Noun} \mid \texttt{NP}] \cdot \textbf{Pr}[\texttt{the} \mid \texttt{Det}] \cdot \\ &\quad \textbf{Pr}[\texttt{sandwich} \mid \texttt{Noun}]\end{aligned}$$

Fair enough. Those are a lot of expressions multiplied together, which will probably produce a small number. But what does it equate, exactly?

$$\begin{aligned}
\textbf{Pr}[\texttt{ROOT}] &= 1 \\
\textbf{Pr}[\texttt{S .}|\texttt{ROOT}] &= 1/3 \\
\textbf{Pr}[\texttt{NP VP}|\texttt{S}] &= 1 \\
\textbf{Pr}[\texttt{Det Noun}|\texttt{NP}] &= 1/2 \\
\textbf{Pr}[\texttt{the}|\texttt{Det}] &= 1/3 \\
\textbf{Pr}[\texttt{president}|\texttt{Noun}] &= 1/6 \\
\textbf{Pr}[\texttt{Verb NP}|\texttt{VP}] &= 1 \\
\textbf{Pr}[\texttt{ate}|\texttt{Verb}] &= 1/5 \\
\textbf{Pr}[\texttt{Det Noun}|\texttt{NP}] &= 1/2 \\
\textbf{Pr}[\texttt{the}|\texttt{Det}] &= 1/3 \\
\textbf{Pr}[\texttt{sandwich}|\texttt{Noun}] &= 1/6
\end{aligned}$$

$\implies$ $\textbf{Pr}[\text{this sentence}] = 5.144032922 \times 10^{-5}$

Let's call this number p, for simplicity purposes.

Using Bayes' Rule, we know that the probability that this is the best parse is equal to the probability that this sentence occurs, multiplied with the probability that we get the best parse *conditioned* on the event of this sentence occurring. That is,

$$\texttt{p(best\_parse)} = \texttt{p(best\_parse|sentence)} \cdot \texttt{p(sentence)}$$

We know that `p(best_parse|sentence)` $= 1$, and we just calculated p. Therefore,

$$\begin{aligned}
\texttt{p(best\_parse)} &= 1 \cdot \textbf{Pr}[\text{this sentence}] \\
&= 5.144032922 \times 10^{-5}
\end{aligned}$$

Now let's think about why `p(sentence)` is equivalent to `p(best_parse)`.

We can again use Bayes' Rule to mathematically argue this. But we can also think intuitively. `p(best_parse)` is the probability that the given parse is the best possible parse. If we know that there is *one* best parse for the specific generated sentence, it *must* be that the probability that this parse is the best parse of *all* best parses for *all* possible sentences is equivalent to the probability of generating this sentence.

Finally, we can think about why `p(best_parse|sentence)` $= 1$. Using the same arguments as above, we know that there is only one possible "best" parse of the sentence. Therefore, the probability that the given parse is equivalent to the *only* best parse is 100%.

We can then check our reasoning:

```
$ ./parse -c -g grammar
the president ate the sandwich .
```

and notice that there indeed is only 1 possible parse of the given sentence.

(ii) Looking at the possibilities from `grammar`, we see that there is actually one other parse that could exist:

```
(ROOT (S (NP (NP (Det every)
                 (Noun sandwich))
             (PP (Prep with)
                (NP (NP (Det a)
                        (Noun pickle))
                    (PP (Prep on)
                        (NP (Det the)
                            (Noun floor))))))
        (VP (Verb wanted)
            (NP (Det a)
                (Noun president))))
    .)
```

Now looking at the probabilities from `grammar` we see that the event that "on the floor" is applied to "a pickle" is equally as likely as the event that "on the floor" is applied to "every sandwich with a pickle".

Because there are only two possible events, and their probabilities are equal, it must be that the probability of each one happening is exactly 0.5.

(iii) First, we calculate the log-probability of the two sentences

$$\begin{aligned} -\text{log-probability} &= -\log_2 p(s_1) - \log_2 p(s_2) \\ &= -\log_2\left(5.144032922 \times 10^-5\right) - \log_2\left(1.240362877 \times 10^-9\right) \\ &= 43.8333311993 \end{aligned}$$

where s_1 and s_2 are the events that the first sentence and second sentence occur, respectively.

Next, we can divide the negative log-probability by 18, the number of words. The resulting cross-entropy is 2.435185067.

(iv) We can calculate perplexity by raising 2 to the power of the cross-entropy.

$$\begin{aligned} \text{perplexity} &= 2^{2.435185067} \\ &= 5.4083370619 \end{aligned}$$

(v) The sentence "`the sandwich ate .`" is not a sentence that can occur using the grammar `grammar`. That is,

$$\mathbf{Pr}[\texttt{the sandwich ate .}] = 0$$

Thus, taking the log-probability results in taking the negative logarithm of 0, which is undefined but approaches infinity.

(e) Calculating the entropy of `grammar2`:

```
$ ./randsent grammar2 1000 | ./parse -P -g grammar2 | ./prettyprint
```

```
# cross-entropy = 2.192609841 bits = -(-24335.77663 log-prob.  / 11099 words)
```

Calculating the entropy of `grammar3`:

```
$ ./randsent grammar3 1000 | ./parse -P -g grammar3 | ./prettyprint
```

```
# cross-entropy = 2.730945129 bits = -(-58756.28445 log-prob.  / 21515 words)
```

The entropy of `grammar3` is larger than the entropy of `grammar2`. This is because `grammar3` is able to generate a wider variety of sentences, which leads to a more "creative" corpus.

Calculating the entropy of `grammar`:

```
$ ./randsent grammar3 1000 | ./parse -P -g grammar3 | ./prettyprint
```

The problem with calculating the entropy of `grammar` is that the rules in `grammar` allow for really *really* long sentences, which take a very long time to generate. We can look at the possible rules for `NP` in `grammar`:

```
1  NP  Det Noun
1  NP  NP PP
```

It is clear that each rule has a 1/2 probability of being executed. If we calculate the probability that a sentence generated with `grammar` terminates, we actually see that it is exactly 1.

$$\begin{aligned}
\mathbf{Pr}[\text{sentence terminates}] &= p \\
p &= \left(1 - \frac{1}{2}\right) \cdot 1 + \frac{1}{2} \cdot p^2 \\
&= \frac{1}{2} + \frac{1}{2} \cdot p^2 \\
\frac{1}{2}p^2 - p + \frac{1}{2} &= 0 \\
p &= 1
\end{aligned}$$

However, note that increasing the probability of `NP` $\rightarrow$ `NP PP` even ever so slightly would cause a probability p such that $0 < p < 1$. That is, the probability of a generated sentence running on forever would non-zero. And because of this, `grammar` produces some really long sentences, which makes calculating the entropy a bit difficult (and slow).

(f) To compare the entropy of a corpus from `grammar2` to the grammar rules `grammar`, `grammar2`, and `grammar3`, it would be a good idea to keep the actual corpus constant. Thus, we can first generate a text file via `grammar2`, and then feed it into `parse` using `grammar`, `grammar2`, and `grammar3`.

First, generating the corpus:

```
$ ./randsent grammar2 1000 > corpus2
```

Next, feeding in the corpus to each `parse`, using each grammar:

```
$ ./parse -P -g grammar < corpus2 | ./prettyprint
$ ./parse -P -g grammar2 < corpus2 | ./prettyprint
$ ./parse -P -g grammar3 < corpus2 | ./prettyprint
```

Finally, the outputs, respectively:

```
# cross-entropy = 2.447528595 bits = -(-26467.57422 log-prob.  / 10814 words)
# cross-entropy = 2.185994451 bits = -(-23639.344 log-prob.  / 10814 words)
# cross-entropy = 2.598891774 bits = -(-28104.41565 log-prob.  / 10814 words)
```

The prediction is correct; `grammar2` on average predicts this corpus better than `grammar` or `grammar3` does.

The reasoning for this could be that using the rules from `grammar` and `grammar3`, `parse` expects to see the sentences generated via `grammar2` less frequently than if using the rules from `grammar2` itself.

7 Extensions

(a) We can use a new symbol `Det_vo` to represent a determiner that can only be used before a word beginning with a vowel. In fact, there is only one word that `Det_vo` turns into: `an`.

Note that in the English grammar, only a noun, adjective, or adverb can immediately follow a determiner. Thus, we also need to define the symbols `Noun_vo`, `Adj_vo`, and `Adv_vo` to represent nouns, adjectives, and adverbs, respectively, that begin with vowels.

To make the grammar file `grammar7` slightly easier to read, we can define `NOUN` to represent *any* type of noun. That its, `NOUN` can turn into either `Noun` *or* `Noun_vo`.

Similarly, we can define `ADJ` to represent any type of adjective.

Note that the new rules that have been added in the grammar for this question have comments along the lines of " `# this rule is for (7a)` ".

(b) To solve this problem, we can use the symbols in the below chart to represent verbs, sentences, verb phrases, and noun clauses in different tenses:

	Past Tense	Present Tense	3rd-personal Singular Form
Transitive Verb	V_tr_ed	V_tr_ori	V_tr_s
Intransitive Verb	V_in_ed	V_in_ori	V_in_s
Sentence	S_pas	S_pre	
Verb Phrase	VP_pas	VP_pre	VPs_pre
Noun Clause	NC_pas	NC_pre	

Each of grammar rules that was implemented before this step is in the past tense. These symbols were changed for less ambiguity.

A yes-no question is composed of an auxiliary word, a sentence whose verb is in its original form, and a question mark at the end. So all we need to do to implement a yes-no question is add the following rule:

$$\texttt{ROOT} \rightarrow \texttt{Aux S\_ori ?}$$

along with a series of rules to expand `S_ori`.

Note that the new rules that have been added in the grammar for this question have comments along the lines of " `# this rule is for (7b)` ".

8 English