This executable notebook will help you complete Pset 5.

If you haven't used Colab before, it's very similar to Jupyter / IPython / R Notebooks: cells containing Python code can be interactively run, and their outputs will be interpolated into this document. If you haven't used any such software before, we recommend taking a quick tour of Colab.

Now, a few Colab-specific things to note about execution before we get started:

- Google offers free compute (including GPU compute!) on this notebook, but *only for a limited time*. Your session will be automatically closed after 12 hours. That means you'll want to finish within 12 hours of starting, or make sure to save your intermediate work (see the next bullet).
- You can save and write files from this notebook, but they are *not guaranteed to persist*. For this reason, we'll mount a Google Drive account and write to that Drive when any files need to be kept permanently (e.g. model checkpoints, surprisal data, etc.).
- You should keep this tab open until you're completely finished with the notebook. If you close the tab, your session will be marked as "Idle" and may be terminated.

Getting started

First, make a copy of this notebook so you can make your own changes. Click *File -> Save a copy in Drive*.

What you need to do

Read through this notebook and execute each cell in sequence, making modifications and adding code where necessary. You should execute all of the code as instructed, and make sure to write code or textual responses wherever the text **TODO** shows up in text and code cells.

When you're finished, choose *File -> Download .ipynb*. You will upload this `.ipynb` file as part of your submission.

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-
packages (3.9.1)
Requirement already satisfied: click in
/usr/local/lib/python3.11/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in
/usr/local/lib/python3.11/dist-packages (from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-
packages (from nltk) (4.67.1)
```

Writing context-free grammars

This is an exercise in writing context-free grammars (CFGs) to capture generalizations about natural language syntax. You can use an automatic parser available in `NLTK` to check whether your grammar accounts for the key generalizations. To get you going, consider the following grammar that we covered in the CFG lectures:

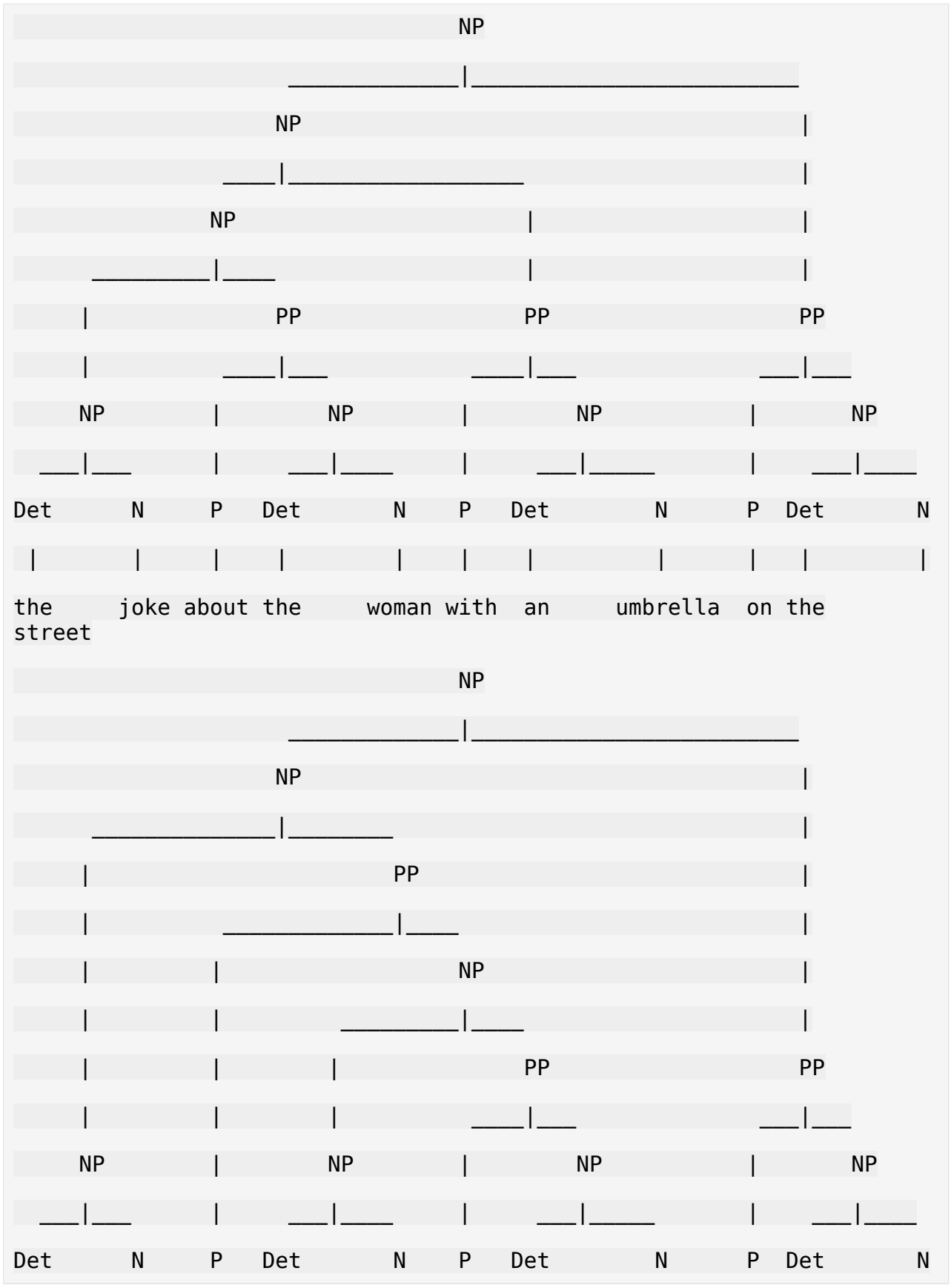
```
NP -> Det N
NP -> NP PP
PP -> P NP
Det -> a
Det -> an
N -> joke
N -> woman
N -> umbrella
N -> street
P -> about
P -> with
P -> on
```

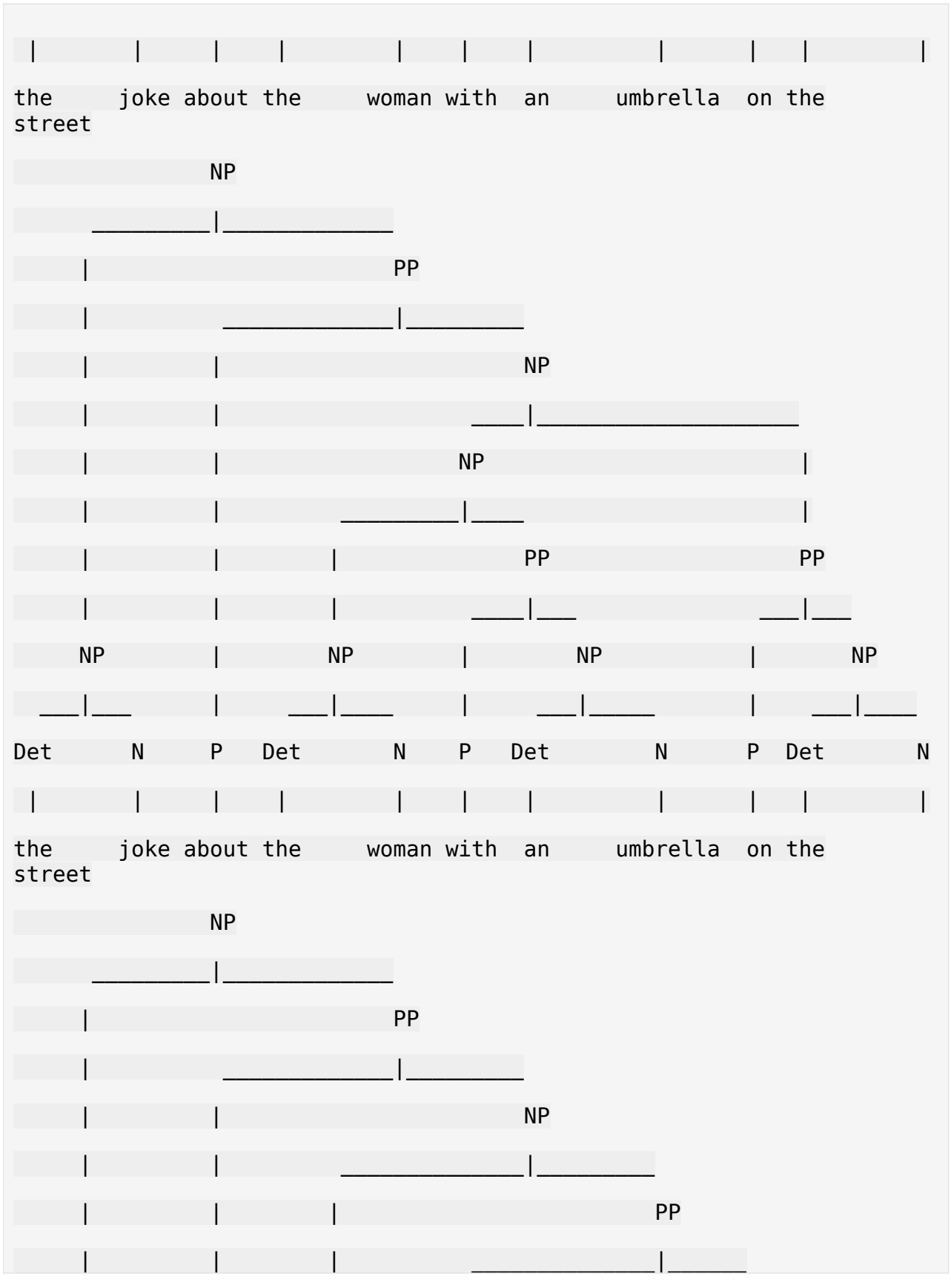
Running the following code will parse the string *a joke about the woman with an umbrella on the street* with start symbol (i.e., goal category) `NP`, generating the five parses that we saw in the CFG lecture notes. Note that `NLTK`'s `CFG.fromstring()` function takes the left-hand-side of the first rule listed as the goal category, and allows multiple rewrites for a single category to be expressed with a disjunction on the right-hand side of a single rule, so that the rule

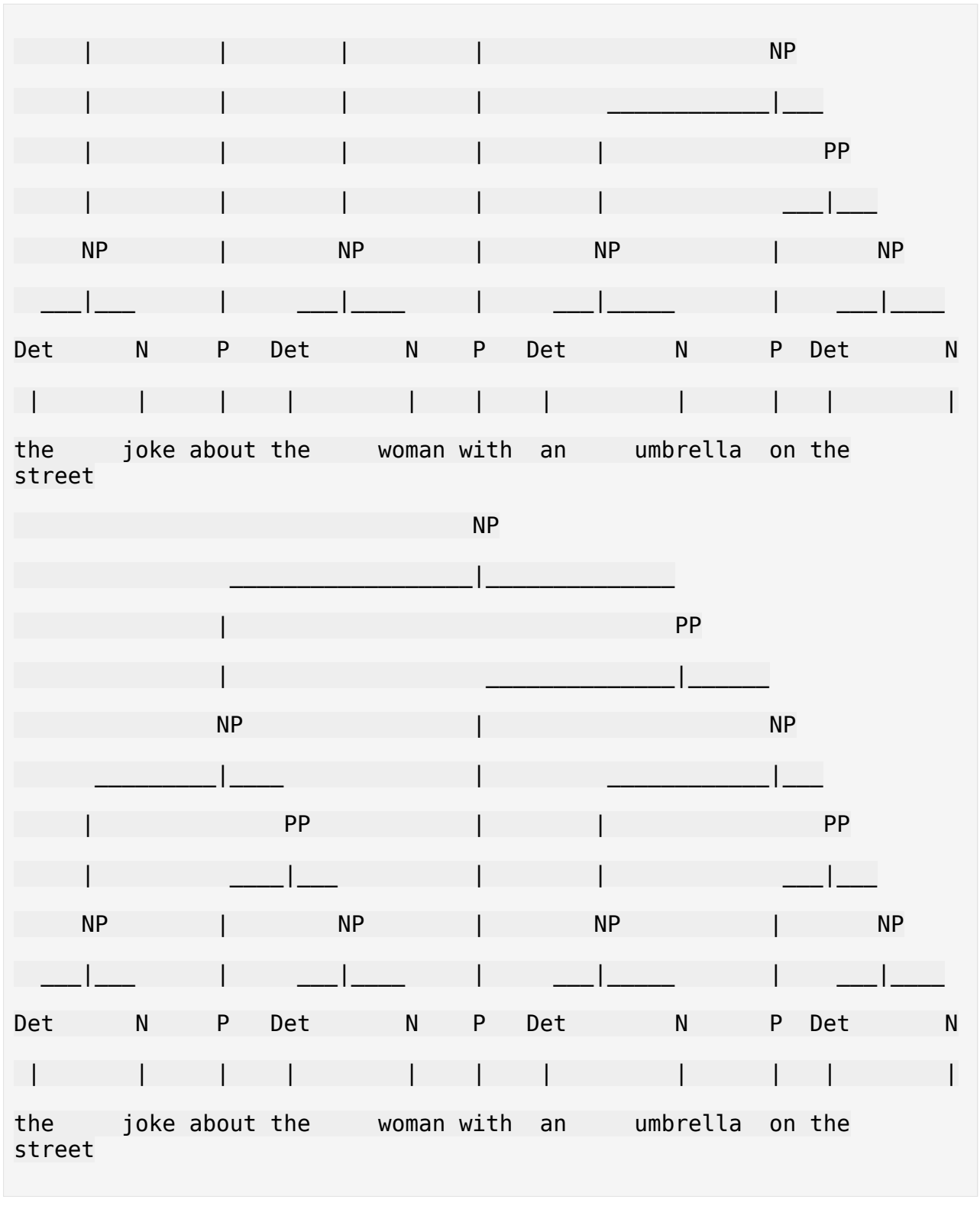
```
X -> Y1 ... Ym | Z1 ... Zn
```

is shorthand for the two rules $X \rightarrow Y_1 \ldots Y_m$ and $X \rightarrow Z_1 \ldots Z_n$.

```
import nltk
from nltk import Nonterminal, nonterminals, Production, CFG

grammar = CFG.fromstring("""
NP -> Det N | NP PP
PP -> P NP
Det -> 'the' | 'a' | 'an'
N -> 'joke' |'woman' | 'umbrella' | 'street'
P -> 'about' | 'with' | 'on'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
sentence = "the joke about the woman with an umbrella on the
street".split()
for tree in parser.parse(sentence):
    tree.pretty_print()
```


NP
NP
NP
NP
PP
PP
PP
NP
NP
NP
NP
Det N P Det N P Det N P Det N
the joke about the woman with an umbrella on the street
NP
NP
PP
NP
NP
PP
PP
NP
NP
NP
NP
Det N P Det N P Det N P Det N


the joke about the woman with an umbrella on the street
NP
PP
NP
NP
NP
PP
PP
NP
NP
NP
NP
Det N P Det N P Det N P Det N
the joke about the woman with an umbrella on the street
NP
PP
NP
PP


NP
PP
NP NP NP NP
Det N P Det N P Det N P Det N
the joke about the woman with an umbrella on the street
NP
PP
NP NP
PP PP
NP NP NP NP
Det N P Det N P Det N P Det N
the joke about the woman with an umbrella on the street

CFGs and ambiguity, part 1.

Write a context-free grammar that will capture the structural ambiguity in the sentence *They are flying planes*. Your grammar should respect the facts that (i) an `NP` should be substitutable with a pronoun given the right context; and (ii) a verb and an immediately following `NP` can combine to form a `VP`. You can check your work with the `NLTK` parser to make sure that your grammar behaves the way you think it will behave.

```
grammar = CFG.fromstring("""
S -> NP VP
NP -> 'they' | 'planes'
VP -> V VP | V ADJ NP
V -> 'are' | 'flying'
ADJ -> 'flying'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
sentence1 = ['they','are','flying','planes']
for tree in parser.parse(sentence1):
    tree.pretty_print()
```

```
          S
  ________|____
 |             VP
 |     ________|______
 NP   V       ADJ     NP
 |    |        |      |
they are     flying planes
```

CFGs and ambiguity, part 2.

Extend your grammar to capture the structural ambiguity in the sentence *Flying planes can be dangerous.* (**Hint:** a non-finite `VP` can serve as the subject of an English sentence, such as in the sentences *To err is human* or *Defeated by the Miami Heat is not how I expected the Milwaukee Bucks to finish in the NBA playoffs*, and it is OK to use a unary rewrite rule with a right-hand-side element that is a phrasal category. See SLP3 Section 12.3.1 for an example of this, though it is a different unary rewrite than you would use for this problem.)

```
grammar = CFG.fromstring("""
S -> NP VP
NP -> 'they' | 'planes' | VP | ADJ NP
VP -> V VP | V ADJ NP | V V ADJ | 'flying' NP
V -> 'are' | 'flying' | 'can' | 'be'
ADJ -> 'flying' | 'dangerous'
""")
parser = nltk.parse.BottomUpChartParser(grammar)
for tree in parser.parse(sentence1):
   tree.pretty_print()
```

```
sentence2 = ['flying','planes','can','be','dangerous']
for tree in parser.parse(sentence2):
   tree.pretty_print()
```

```
          S
  ________|____
 |             VP
 |     ________|______
 NP   V       ADJ     NP
 |    |        |      |
they are     flying planes
```

```
          S
  ________|____
 |             VP
 |     ________|_____
 |    |              VP
 |    |         _____|____
 NP   V        |          NP
 |    |        |          |
they are     flying     planes
```

```
                    S
          __________|___
         NP             VP
   ______|____        ___|______
 ADJ          NP    V   V      ADJ
  |           |     |   |       |
flying     planes can  be dangerous
```

```
                    S
          __________|___
         NP             |
         |              |
         VP             VP
   ______|____        ___|______
  |           NP    V   V      ADJ
  |           |     |   |       |
flying     planes can  be dangerous
```

CFGs and ambiguity, part 3.

The ambiguity of the preceding sentence is eliminated if you change *can be* to either *is* or *are*. Why? Modify your grammar so that it captures this disambiguation effect.

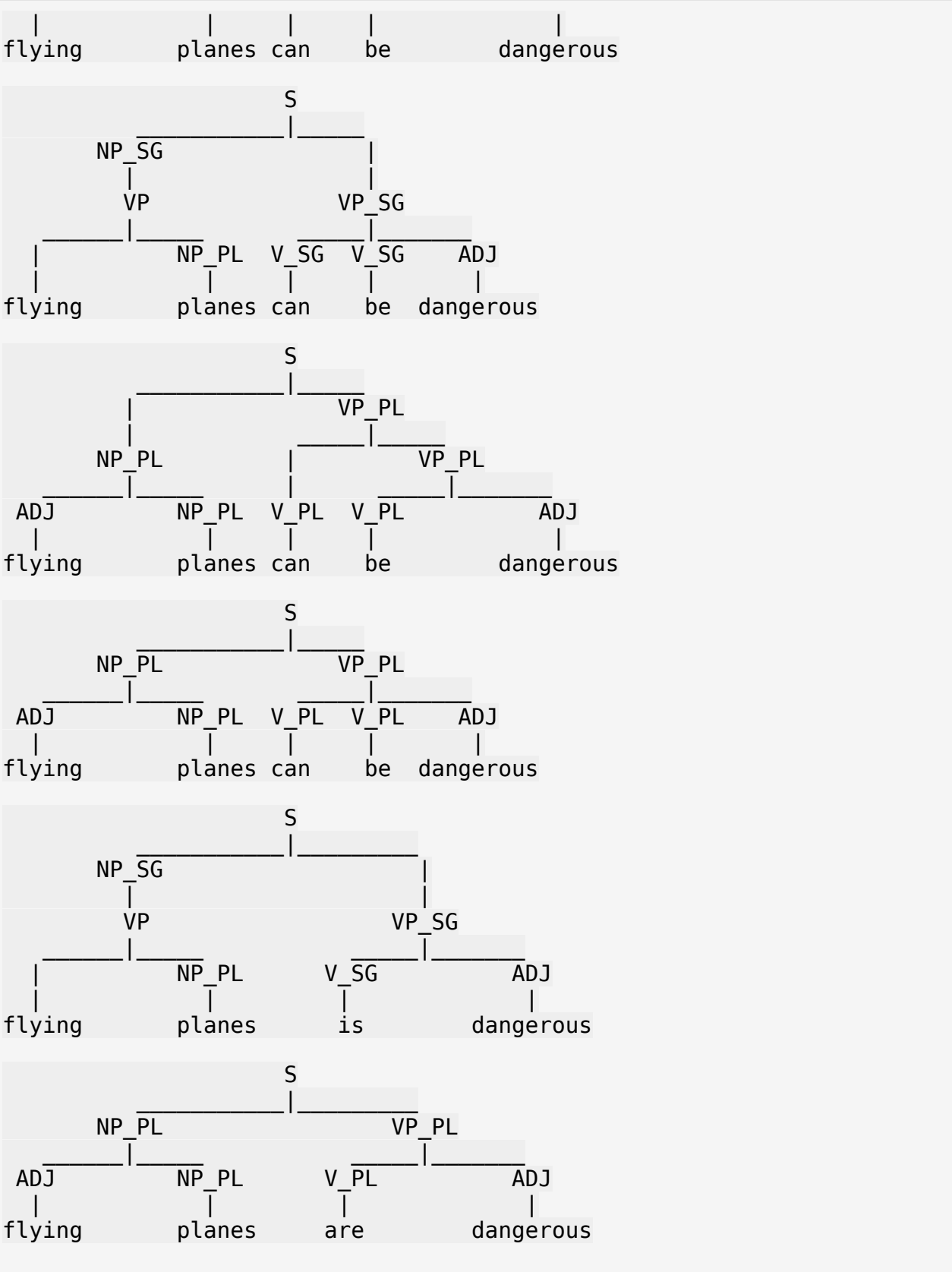
Why this works: Our grammar enforces **subject-verb number** agreement through separate singular (SG) and plural (PL) categories. The modal "can" doesn't have number restrictions, but "is" and "are" do, forcing grammatical agreement that eliminates one of the two possible interpretations.

```
grammar = CFG.fromstring("""
S -> NP_SG VP_SG | NP_PL VP_PL
NP_SG -> VP
NP_PL -> 'they' | 'planes' | ADJ NP_PL
VP_SG -> V_SG VP_SG | V_SG ADJ NP_PL | V_SG V_SG ADJ | V_SG ADJ
VP_PL -> V_PL VP_PL | V_PL ADJ NP_PL | V_PL V_PL ADJ | V_PL ADJ | V_PL
VP
VP -> 'flying' NP_PL
V_SG -> 'flying' | 'can' | 'be' | 'is'
V_PL -> 'are' | 'flying' | 'can' | 'be'
ADJ -> 'flying' | 'dangerous'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
for tree in parser.parse(sentence1):
    tree.pretty_print()
for tree in parser.parse(sentence2):
    tree.pretty_print()
sentence3 = ['flying','planes','is','dangerous']
for tree in parser.parse(sentence3):
    tree.pretty_print()
sentence4 = ['flying','planes','are','dangerous']
for tree in parser.parse(sentence4):
    tree.pretty_print()
```

```
             S
   __________|____
  |            VP_PL
  |       _____|______
NP_PL V_PL       ADJ   NP_PL
  |    |          |      |
 they are       flying planes
```

```
             S
   __________|____
  |            VP_PL
  |       _____|_____
  |      |           VP
  |      |      _____|____
NP_PL V_PL      |        NP_PL
  |    |        |          |
 they are      flying     planes
```

```
                     S
            _________|_____
       NP_SG             VP_SG
         |            _____|_____
         VP          |         VP_SG
   ______|_____      |     ____|_____
  |           NP_PL  V_SG  V_SG      ADJ
```


flying planes can be dangerous
S
NP_SG VP
VP_SG
NP_PL V_SG V_SG ADJ
flying planes can be dangerous
S
VP_PL
NP_PL VP_PL
ADJ NP_PL V_PL V_PL ADJ
flying planes can be dangerous
S
NP_PL VP_PL
ADJ NP_PL V_PL V_PL ADJ
flying planes can be dangerous
S
NP_SG VP
VP_SG
NP_PL V_SG ADJ
flying planes is dangerous
S
NP_PL VP_PL
ADJ NP_PL V_PL ADJ
flying planes are dangerous

Problem 2: Adding argument structure and unbounded dependencies to a context-free grammar

```
import nltk
from nltk import Nonterminal, nonterminals, Production, CFG

grammar = CFG.fromstring("""
S -> NP VP
NP -> Det N | Pronoun
VP -> V
VP -> V NP
VP -> V SBAR
SBAR -> WHNP S
SBAR -> COMP S
COMP -> 'that' |
WHNP -> 'who' | 'what'
Det -> 'the' | 'a' | 'an' | 'my' | 'your' | 'her' | 'his' | 'their'
N -> 'joke' | 'women' | 'street' | 'apple'
Pronoun -> 'I' | 'you' | 'she' | 'he' | 'they'
V -> 'slept' | 'devoured' | 'know' | 'said' | 'know'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
good_sentences = ["I devoured the apple".split(),
             "I said you slept".split(),
             "you know what I devoured".split(),
             "you know who slept".split(),
             "the women know who I said devoured the apple".split()
             ]

bad_sentences = [
             "I slept that you said".split(),
             "the women devoured".split(),
             "I know what you said the joke".split()
]

def check_good_sentence(sentence):
  parses = parser.parse(sentence)
  num_parses = sum(1 for dummy in parser.parse(sentence))
  if num_parses == 0:
    print("Incorrectly rejected '" + " ".join(sentence) + "'!")
  for tree in parser.parse(sentence):
    tree.pretty_print()

def check_bad_sentence(sentence):
  parses = parser.parse(sentence)
  num_parses = sum(1 for dummy in parser.parse(sentence))
  if num_parses > 0:
    print("Incorrectly accepted '" + " ".join(sentence) + "'!")
```

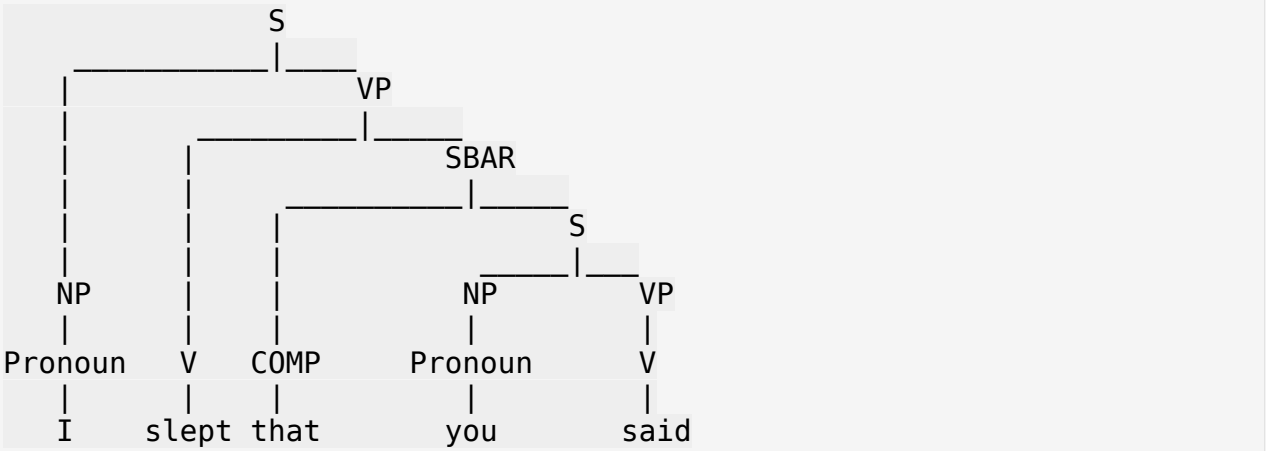
```
  for tree in parser.parse(sentence):
    tree.pretty_print()

for sentence in good_sentences:
  check_good_sentence(sentence)

for sentence in bad_sentences:
  check_bad_sentence(sentence)
```

```
                  S
    ______________|___
   |                  VP
   |           _______|___
   NP         |           NP
   |          |        ___|____
Pronoun       V      Det       N
   |          |       |        |
   I       devoured   the     apple
```

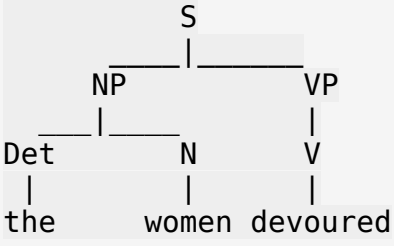
```
               S
    ___________|____
   |                VP
   |          ______|_____
   |         |            SBAR
   |         |      ______|_____
   |         |     |            S
   |         |     |        ____|____
   NP        |     |       NP        VP
   |         |     |       |         |
Pronoun      V   COMP    Pronoun     V
   |         |     |       |         |
   I       said ...       you      slept
```

```
               S
    ___________|____
   |                VP
   |          ______|_____
   |         |            SBAR
   |         |      ______|_____
   |         |     |            S
   |         |     |        ____|_____
   NP        |     |       NP         VP
   |         |     |       |          |
Pronoun      V   WHNP    Pronoun      V
   |         |     |       |          |
  you      know what      I        devoured
```

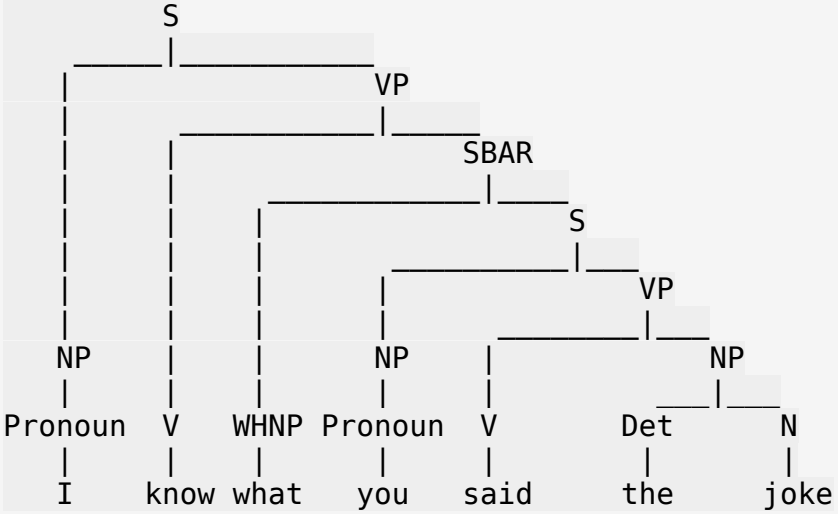
```
Incorrectly rejected 'you know who slept'!
Incorrectly rejected 'the women know who I said devoured the apple'!
Incorrectly accepted 'I slept that you said'!
```



Incorrectly accepted 'the women devoured'!



Incorrectly accepted 'I know what you said the joke'!



As you see, the above grammar gives correct parses for sentences like: \

(1) I devoured the apple \

(2) I said you slept \

(3) you know what I devoured \

However, it will incorrectly accept sentences like: \

(4) *I slept that you said \

(5) *the women devoured \

(6) *I know what you said the joke \

and incorrectly reject sentences like: \

(7) you know who slept \

(8) the women know who I said devoured the apple

Part 1: argument structure.

Revise the grammar so that it correctly accounts for the different argument structures of the different verbs.

At the end of this exercise, your grammar should yield the following behavior: Your grammar should now **correctly reject** (5) and (6): \

(5) *the women devoured \

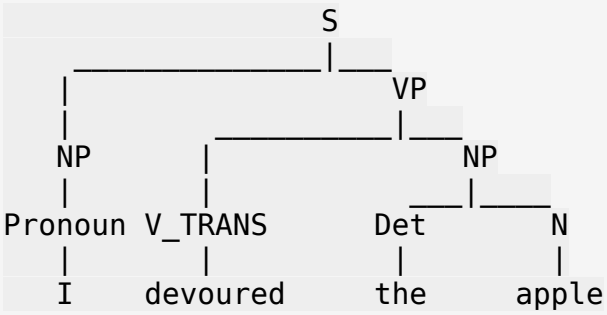
(6) *I know what you said the joke \

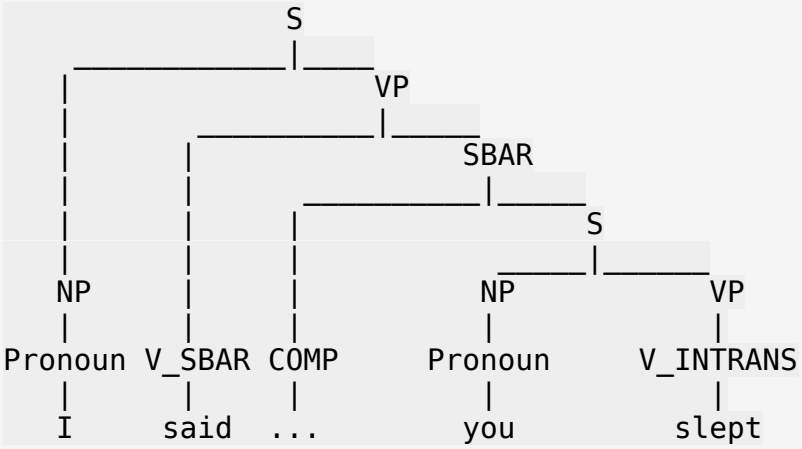
but **incorrectly reject** (3):

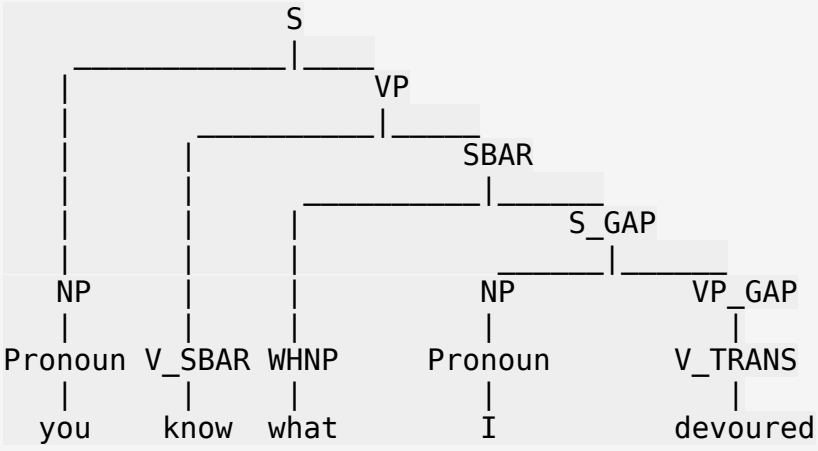
(3) you know what I devoured

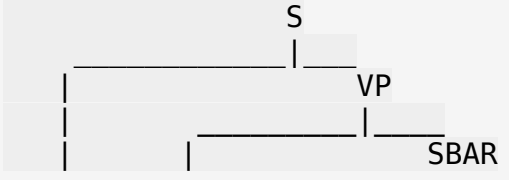
```
grammar = CFG.fromstring("""
S -> NP VP
NP -> Det N | Pronoun
VP -> V_INTRANS
VP -> V_TRANS NP
VP -> V_SBAR SBAR
SBAR -> WHNP S_GAP
SBAR -> COMP S
SBAR_GAP -> S_GAP
S_GAP -> NP VP_GAP
S_GAP -> VP_GAP
VP_GAP -> V_INTRANS
VP_GAP -> V_TRANS
VP_GAP -> V_TRANS NP
VP_GAP -> V_SBAR SBAR
VP_GAP -> V_SBAR SBAR_GAP
COMP -> 'that' |
WHNP -> 'who' | 'what'
Det -> 'the' | 'a' | 'an' | 'my' | 'your' | 'her' | 'his' | 'their'
N -> 'joke' | 'women' | 'street' | 'apple'
Pronoun -> 'I' | 'you' | 'she' | 'he' | 'they'
V_INTRANS -> 'slept'
V_TRANS -> 'devoured'
V_SBAR -> 'know' | 'said'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
```

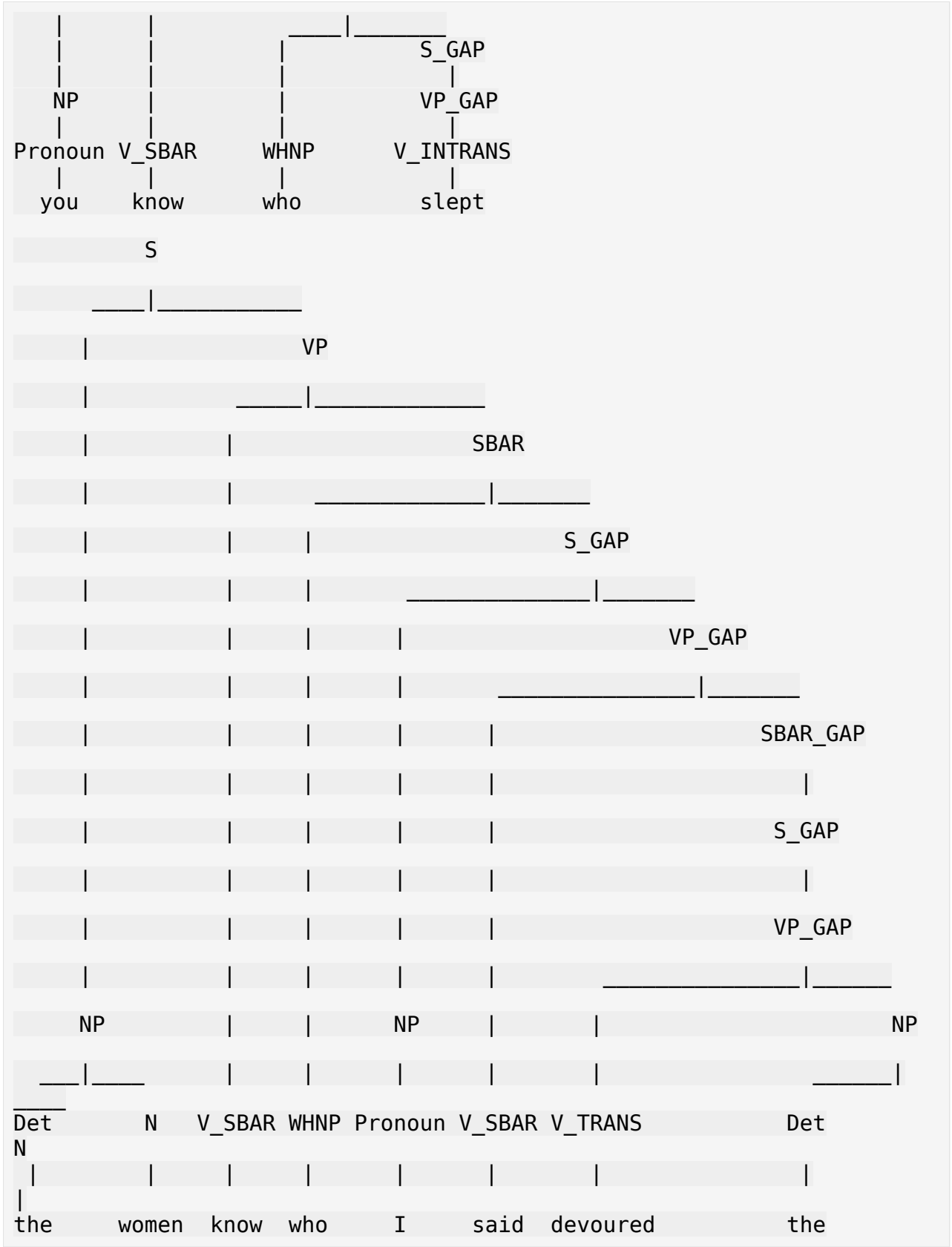
```
for sentence in good_sentences:
  check_good_sentence(sentence)

for sentence in bad_sentences:
  check_bad_sentence(sentence)
```










Pronoun V_SBAR WHNP V_INTRANS
you know who slept
S
VP
SBAR
S_GAP
VP_GAP
SBAR_GAP
S_GAP
VP_GAP
NP NP NP
Det N V_SBAR WHNP Pronoun V_SBAR V_TRANS Det N
the women know who I said devoured the

```
apple
```

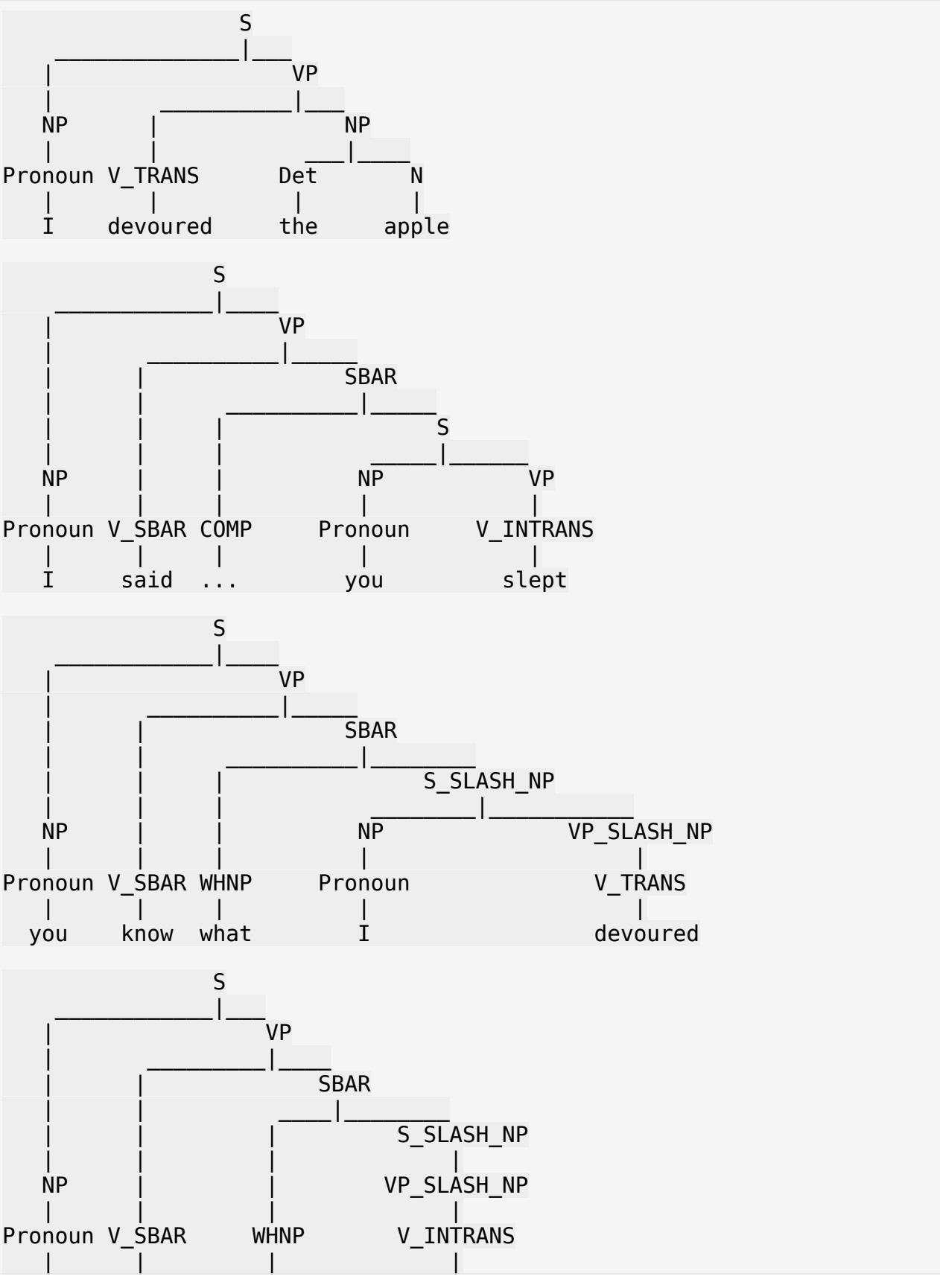
Want a challenge? (optional)

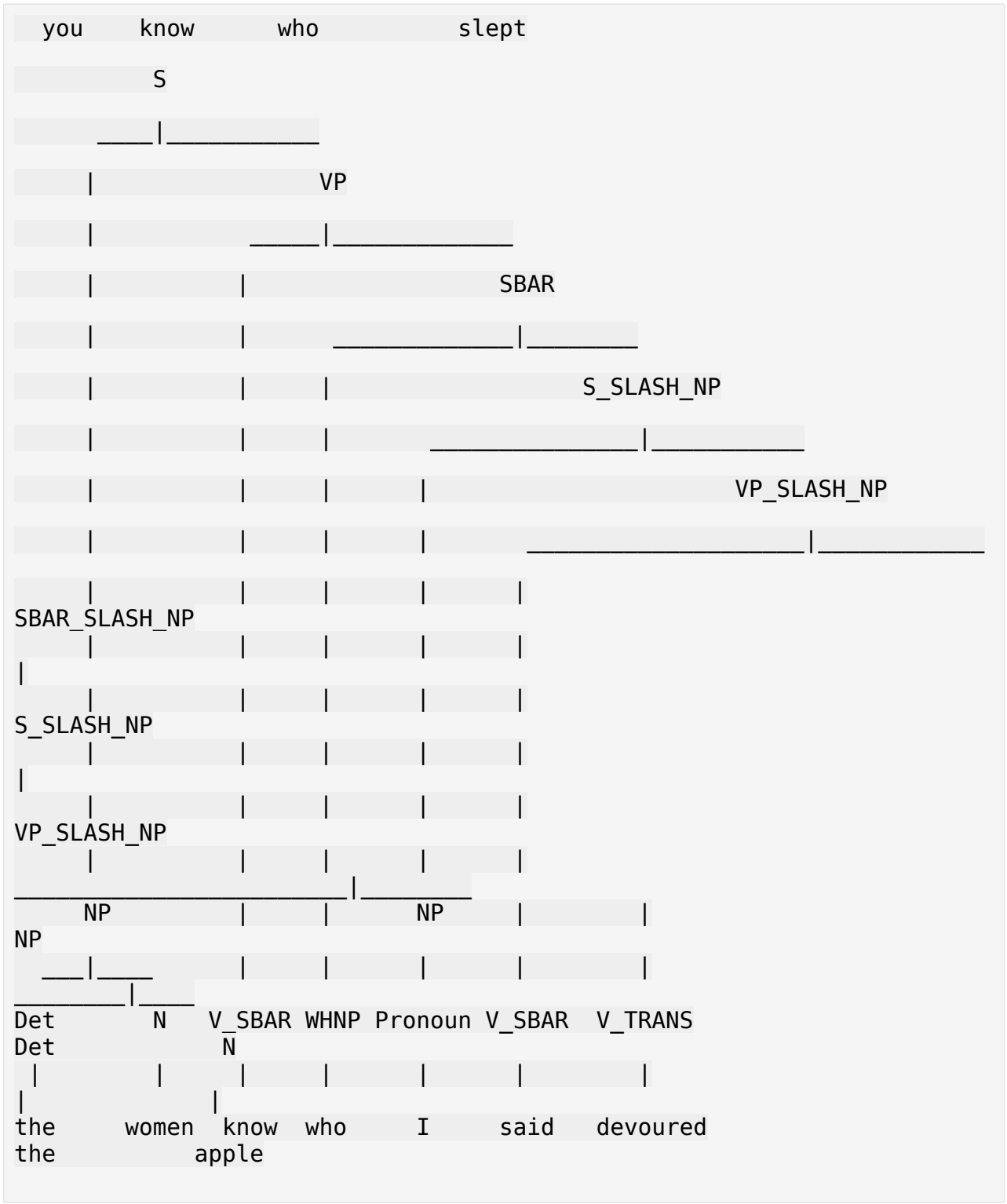
Implement meta-rules using `S`, `NP`, `VP`, and `SBAR` as your **basic categories**, and all the non-terminal rewrites as your **basic rules**, to add a set of new **derived categories** and **derived rules to your grammar**.

By the end of this exercise, your grammar should now **correctly accept and reject all the above eight examples.**

(This challenge is not worth any points, but we will give you feedback on any solution you offer.)

```
grammar = CFG.fromstring("""
S -> NP VP
NP -> Det N | Pronoun
VP -> V_INTRANS
VP -> V_TRANS NP
VP -> V_SBAR SBAR
SBAR -> WHNP S
SBAR -> WHNP S_SLASH_NP
SBAR -> COMP S

S_SLASH_NP -> VP_SLASH_NP
S_SLASH_NP -> NP VP_SLASH_NP
VP_SLASH_NP -> V_INTRANS
VP_SLASH_NP -> V_TRANS
VP_SLASH_NP -> V_TRANS NP
VP_SLASH_NP -> V_SBAR SBAR
VP_SLASH_NP -> V_SBAR SBAR_SLASH_NP
SBAR_SLASH_NP -> WHNP S_SLASH_NP
SBAR_SLASH_NP -> S_SLASH_NP
NP_SLASH_NP ->

COMP -> 'that' |
WHNP -> 'who' | 'what'
Det -> 'the' | 'a' | 'an' | 'my' | 'your' | 'her' | 'his' | 'their'
N -> 'joke' | 'women' | 'street' | 'apple'
Pronoun -> 'I' | 'you' | 'she' | 'he' | 'they'
V_INTRANS -> 'slept'
V_TRANS -> 'devoured'
V_SBAR -> 'know' | 'said'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
for sentence in good_sentences:
    check_good_sentence(sentence)
for sentence in bad_sentences:
    check_bad_sentence(sentence)
```


S
VP
NP
NP
Pronoun V_TRANS Det N
I devoured the apple
S
VP
SBAR
S
NP VP
Pronoun V_SBAR COMP Pronoun V_INTRANS
I said ... you slept
S
VP
SBAR
S_SLASH_NP
NP VP_SLASH_NP
Pronoun V_SBAR WHNP Pronoun V_TRANS
you know what I devoured
S
VP
SBAR
S_SLASH_NP
VP_SLASH_NP
NP
Pronoun V_SBAR WHNP V_INTRANS


you know who slept
S
VP
SBAR
S_SLASH_NP
VP_SLASH_NP
SBAR_SLASH_NP
S_SLASH_NP
VP_SLASH_NP
NP NP NP
Det N V_SBAR WHNP Pronoun V_SBAR V_TRANS
Det N
the women know who I said devoured
the apple