Homework 5

Spencer Hann

CS 562 | Winter 2019

Part 1: Collapse Unary Productions

Deliverable 1.1

```
    def collapse_unary(self, join_char=CU_JOIN_CHAR):
        # daughter is terminal node
        if type(self.daughters[0]) == str: return
        # daughter is pre-terminal node
        if type(self.daughters[0].daughters[0]) == str: return

        while len(self.daughters) == 1 and self.label != "TOP":
            self.label += join_char + self.daughters[0].label
            self.daughters = self.daughters[0].daughters

        for daughter in self.daughters:
            daughter.collapse_unary()

        return self
```

My code for the `Tree.collapse_unary` function, sans the docstring.

Deliverable 1.3

My approach for this function was relatively straight forward. The problem itself is not as complex as the other two, so that seemed appropriate. I simply do a recursive, top-down, left-to-right, traversal of the tree. When a node is found with a single child, it absorbs that child. The only bugs I ran into had to do with not checking for "`TOP`", which should not be collapsed, and not checking for the `str` data type. The most challenging thing about Part 1 for me was not the function itself, but rather, taking the time to learn the data-structure and read the provided code. Once I had done that, though, it made this function and the next two problems easier.

I swapped deliverables x.*3 and* x.*2 so that the code/bug summary is directly below where the code is displayed, so that it's easier to look back and forth between the code and the code-writeup.*

Deliverable 1.2

```
Trying:
    s = '(TOP (S (VP (TO to) (VP (VB play)))))'
```

```
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
    (TOP
        (S+VP
            (TO to)
            (VP
                (VB play)
            )
        )
    )
ok
Trying:
    s = '(TOP (S (SBAR (VP (TO to) (VP (VB play))))))'
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
    (TOP
        (S+SBAR+VP
            (TO to)
            (VP
                (VB play)
            )
        )
    )
ok
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
```

```
    (TOP
        (S
            (S+VP
                (VBN Turned)
                (ADVP
                    (RB loose)
                )
                (PP
                    (IN in)
                    (NP
                        (NP
                            (NNP Shane)
                            (NNP Longman)
                            (POS 's)
                        )
                        (NN trading)
                        (NN room)
                    )
                )
            )
            (, ,)
            (NP
                (DT the)
                (NN yuppie)
                (NNS dealers)
            )
            (VP
                (AUX do)
                (NP
                    (NP
                        (RB little)
                    )
                    (ADJP
                        (RB right)
                    )
                )
            )
            (. .)
        )
    )
ok
```

Part 2: Chomsky Normal Form

Deliverable 2.1

```
    def chomsky_normal_form(self, markovize_char=MARKOVIZE_CHAR,
                            join_char=CNF_JOIN_CHAR,
                            left_delimiter=CNF_LEFT_DELIMITER,
                            right_delimiter=CNF_RIGHT_DELIMITER):
        for dtr in self.daughters:
```

```
            if type(dtr) == str: continue
            dtr.chomsky_normal_form()

        search_regex = r"{}(.*){}".format(left_delimiter, join_char)
        search_regex = re.compile(search_regex)

        while len(self.daughters) > 2:
            left = self.daughters[-2]
            right = self.daughters[-1]

            del self.daughters[-1]

            rlabel = re.search(search_regex, right.label)
            rlabel = rlabel.group(1) if rlabel else right.label

            self.daughters[-1] = Tree(
                        self.label + markovize_char + left_delimiter +
                        left.label + join_char +
                        rlabel + right_delimiter
            )
            self.daughters[-1].daughters = [left, right]

        return self
```

Deliverable 2.3

This function, for me had the most significant but of the three. When I first wrote it, it seemed simple to me and I was pretty confident I had solved it. However, I had issues with nesting rules. For example, I would see nodes labeled things like `a|<b&a|<c&d>>`, I struggled with a while about, why this would be incorrect, and how to fix it. I like working on this bug, because I had never used `re`'s `group` method before. I read the given function, `Tree.from_string` more closely and looked up the documentation for the `group` function, which it uses, and decided to try to use it to solve this problem. I'm glad that I took the time to do it this way, because I can see how the `group` function used with Python's regular expressions can be a powerful tool in many situations.

Deliverable 2.2

```
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    Tree.from_string(s).collapse_unary().chomsky_normal_form()
Expecting:
    (TOP
        (S
            (S+VP
```

```
                (VBN Turned)
                (S+VP|<ADVP&PP>
                    (ADVP
                        (RB loose)
                    )
                    (PP
                        (IN in)
                        (NP
                            (NP
                                (NNP Shane)
                                (NP|<NNP&POS>
                                    (NNP Longman)
                                    (POS 's)
                                )
                            )
                            (NP|<NN&NN>
                                (NN trading)
                                (NN room)
                            )
                        )
                    )
                )
            )
            (S|<,&NP>
                (, ,)
                (S|<NP&VP>
                    (NP
                        (DT the)
                        (NP|<NN&NNS>
                            (NN yuppie)
                            (NNS dealers)
                        )
                    )
                    (S|<VP&.>
                        (VP
                            (AUX do)
                            (NP
                                (NP
                                    (RB little)
                                )
                                (ADJP
                                    (RB right)
                                )
                            )
                        )
                        (. .)
                    )
                )
            )
        )
    )
ok
```

Part 3: Generate Productions

Deliverable 3.1

```
    def productions(self):
        mothers = list()
        daughters = list()

        self._productions(mothers, daughters)

        return zip(mothers, daughters)

    def _productions(self, mothers, daughters):
        mothers.append(self.label)

        if type(self.daughters[0]) is str: # terminal
            daughters.append(self.daughters)
            return

        if len(self.daughters) == 2:
            daughters.append(
                [
                self.daughters[0] if type(self.daughters[0]) is str \
                    else self.daughters[0].label, \
                self.daughters[1] if type(self.daughters[1]) is str \
                    else self.daughters[1].label
                ]
            )
        else:
            daughters.append(
                [self.daughters[0] if type(self.daughters[0]) is str \
                    else self.daughters[0].label]
                )

        if type(self.daughters[0]) is Tree:
            self.daughters[0]._productions(mothers, daughters)
        if len(self.daughters) == 2 and type(self.daughters[1]) is Tree:
            self.daughters[1]._productions(mothers, daughters)
```

Deliverable 3.3

This function, like the other two, I wrote recursively. However, with this function I used a wrapper, this way I could initialize my lists before entering the recursive function, so that I could operate on the same lists the whole time, rather than constantly returning and merging separate lists which would be far less efficient. I was able to get this function working almost immediately... sort of.

I did have a bug where, when I hit a terminal or pre-terminal node, basically, any time there was only one child, I would add that `str` to the `daughters` list. Otherwise, I would add a `tuple` of `str`'s. The test for this functions uses a `' '.join(daughters)` line to display the output. This works fine when a `tuple` of `"one"` and `"two"` get joined as "`one two`". However, an individual instance of the `str` type, `"this is just one`

`string"` will get joined as `"t h i s i s j u s t o n e s t r i n g"`. So, while all the correct information was there, it was displaying improperly and causing the test to fail. I tried several other solutions, but eventually ended up just wrapping everything, including individual strings, in lists.

Deliverable 3.2

```
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    t = Tree.from_string(s).collapse_unary().chomsky_normal_form()
Expecting nothing
ok
Trying:
    for (mother, daughters) in t.productions():
        print('{: <20} -> {}'.format(mother, ' '.join(daughters)))
Expecting:
    TOP                  -> S
    S                    -> S+VP S|<,&NP>
    S+VP                 -> VBN S+VP|<ADVP&PP>
    VBN                  -> Turned
    S+VP|<ADVP&PP>       -> ADVP PP
    ADVP                 -> RB
    RB                   -> loose
    PP                   -> IN NP
    IN                   -> in
    NP                   -> NP NP|<NN&NN>
    NP                   -> NNP NP|<NNP&POS>
    NNP                  -> Shane
    NP|<NNP&POS>         -> NNP POS
    NNP                  -> Longman
    POS                  -> 's
    NP|<NN&NN>           -> NN NN
    NN                   -> trading
    NN                   -> room
    S|<,&NP>             -> , S|<NP&VP>
    ,                    -> ,
    S|<NP&VP>            -> NP S|<VP&.>
    NP                   -> DT NP|<NN&NNS>
    DT                   -> the
    NP|<NN&NNS>          -> NN NNS
    NN                   -> yuppie
    NNS                  -> dealers
    S|<VP&.>             -> VP .
    VP                   -> AUX NP
    AUX                  -> do
    NP                   -> NP ADJP
    NP                   -> RB
```

```
    RB                  -> little
    ADJP                -> RB
    RB                  -> right
    .                   -> .
ok
```

Full terminal output

To get the terminal output, I changed the "`__main__`" section of the source code at the bottom of the file so that running `python tree.py` with a "verbose" `-v` argument from the command line would set the `doctest.testmod` function's `verbose` parameter to `True`. Otherwise, `doctest.testmod` does not output anything when all tests are passed.

```
if __name__ == '__main__':
    import doctest
    import sys
    if "-v" in sys.argv:
        v = True
    else:
        v = False
    doctest.testmod(verbose=v)
```

Here is the output at the end of the tests:

```
6 items passed all tests:
   2 tests in __main__.Tree.chomsky_normal_form
   9 tests in __main__.Tree.collapse_unary
   5 tests in __main__.Tree.from_stream
   7 tests in __main__.Tree.from_string
   3 tests in __main__.Tree.pretty
   3 tests in __main__.Tree.productions
29 tests in 19 items.
29 passed and 0 failed.
Test passed.
```

And here is the full terminal output:

```
spencer@rBS:~/ohsu/562/homework_5$ python tree.py -v
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    Tree.from_string(s).collapse_unary().chomsky_normal_form()
```

```
Expecting:
    (TOP
        (S
            (S+VP
                (VBN Turned)
                (S+VP|<ADVP&PP>
                    (ADVP
                        (RB loose)
                    )
                    (PP
                        (IN in)
                        (NP
                            (NP
                                (NNP Shane)
                                (NP|<NNP&POS>
                                    (NNP Longman)
                                    (POS 's)
                                )
                            )
                            (NP|<NN&NN>
                                (NN trading)
                                (NN room)
                            )
                        )
                    )
                )
            )
            (S|<,&NP>
                (, ,)
                (S|<NP&VP>
                    (NP
                        (DT the)
                        (NP|<NN&NNS>
                            (NN yuppie)
                            (NNS dealers)
                        )
                    )
                    (S|<VP&.>
                        (VP
                            (AUX do)
                            (NP
                                (NP
                                    (RB little)
                                )
                                (ADJP
                                    (RB right)
                                )
                            )
                        )
                        (. .)
                    )
                )
            )
```

```
        )
    )
ok
Trying:
    s = '(TOP (S (VP (TO to) (VP (VB play)))))'
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
    (TOP
        (S+VP
            (TO to)
            (VP
                (VB play)
            )
        )
    )
ok
Trying:
    s = '(TOP (S (SBAR (VP (TO to) (VP (VB play))))))'
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
    (TOP
        (S+SBAR+VP
            (TO to)
            (VP
                (VB play)
            )
        )
    )
ok
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
```

```
Expecting nothing
ok
Trying:
    t.collapse_unary()
Expecting:
    (TOP
        (S
            (S+VP
                (VBN Turned)
                (ADVP
                    (RB loose)
                )
                (PP
                    (IN in)
                    (NP
                        (NP
                            (NNP Shane)
                            (NNP Longman)
                            (POS 's)
                        )
                        (NN trading)
                        (NN room)
                    )
                )
            )
            (, ,)
            (NP
                (DT the)
                (NN yuppie)
                (NNS dealers)
            )
            (VP
                (AUX do)
                (NP
                    (NP
                        (RB little)
                    )
                    (ADJP
                        (RB right)
                    )
                )
            )
            (. .)
        )
    )
ok
Trying:
    from io import StringIO
Expecting nothing
ok
Trying:
    s = '(ADVP (ADV widely) (CONJ and) (ADV friendly))'
Expecting nothing
```

```
ok
Trying:
    source = StringIO(s.replace(' ', '\n\n\n') + s)
Expecting nothing
ok
Trying:
    (one, two) = Tree.from_stream(source)
Expecting nothing
ok
Trying:
    str(one) == str(two)
Expecting:
    True
ok
Trying:
    s = '(ADVP (ADV widely) (CONJ and) (ADV friendly))'
Expecting nothing
ok
Trying:
    Tree.from_string(s)
Expecting:
    (ADVP
        (ADV widely)
        (CONJ and)
        (ADV friendly)
    )
ok
Trying:
    str(Tree.from_string(s)) == \
    str(Tree.from_string(s.replace(' ', '\n')))
Expecting:
    True
ok
Trying:
    Tree.from_string(s[:-1])
Expecting:
    Traceback (most recent call last):
    ...
    ValueError: End-of-string, need /\)/
ok
Trying:
    Tree.from_string(s[1:])
Expecting:
    Traceback (most recent call last):
    ...
    ValueError: Need /\(/
ok
Trying:
    s_without_head = s[6:-1]
Expecting nothing
ok
Trying:
    Tree.from_string(s_without_head)
```

```
Expecting:
    Traceback (most recent call last):
    ...
    ValueError: String contains 3 trees
ok
Trying:
    s = '(TOP (S (VP (TO to) (VP (VB play)))))'
Expecting nothing
ok
Trying:
    t = Tree.from_string(s)
Expecting nothing
ok
Trying:
    t
Expecting:
    (TOP
        (S
            (VP
                (TO to)
                (VP
                    (VB play)
                )
            )
        )
    )
ok
Trying:
    s = '''(TOP (S (S (VP (VBN Turned) (ADVP (RB loose)) (PP
           (IN in) (NP (NP (NNP Shane) (NNP Longman) (POS 's))
           (NN trading) (NN room))))) (, ,) (NP (DT the)
           (NN yuppie) (NNS dealers)) (VP (AUX do) (NP (NP
           (RB little)) (ADJP (RB right)))) (. .)))'''
Expecting nothing
ok
Trying:
    t = Tree.from_string(s).collapse_unary().chomsky_normal_form()
Expecting nothing
ok
Trying:
    for (mother, daughters) in t.productions():
        print('{: <20} -> {}'.format(mother, ' '.join(daughters)))
Expecting:
    TOP                  -> S
    S                    -> S+VP S|<,&NP>
    S+VP                 -> VBN S+VP|<ADVP&PP>
    VBN                  -> Turned
    S+VP|<ADVP&PP>       -> ADVP PP
    ADVP                 -> RB
    RB                   -> loose
    PP                   -> IN NP
    IN                   -> in
    NP                   -> NP NP|<NN&NN>
```

```
    NP                    -> NNP NP|<NNP&POS>
    NNP                   -> Shane
    NP|<NNP&POS>          -> NNP POS
    NNP                   -> Longman
    POS                   -> 's
    NP|<NN&NN>            -> NN NN
    NN                    -> trading
    NN                    -> room
    S|<,&NP>              -> , S|<NP&VP>
    ,                     -> ,
    S|<NP&VP>             -> NP S|<VP&.>
    NP                    -> DT NP|<NN&NNS>
    DT                    -> the
    NP|<NN&NNS>           -> NN NNS
    NN                    -> yuppie
    NNS                   -> dealers
    S|<VP&.>              -> VP .
    VP                    -> AUX NP
    AUX                   -> do
    NP                    -> NP ADJP
    NP                    -> RB
    RB                    -> little
    ADJP                  -> RB
    RB                    -> right
    .                     -> .
ok
13 items had no tests:
    __main__
    __main__.Tree
    __main__.Tree.__getitem__
    __main__.Tree.__init__
    __main__.Tree.__iter__
    __main__.Tree.__len__
    __main__.Tree.__repr__
    __main__.Tree.__setitem__
    __main__.Tree._productions
    __main__.Tree.append
    __main__.Tree.pop
    __main__.Tree.terminal
    __main__.Tree.unary
6 items passed all tests:
   2 tests in __main__.Tree.chomsky_normal_form
   9 tests in __main__.Tree.collapse_unary
   5 tests in __main__.Tree.from_stream
   7 tests in __main__.Tree.from_string
   3 tests in __main__.Tree.pretty
   3 tests in __main__.Tree.productions
29 tests in 19 items.
29 passed and 0 failed.
Test passed.
spencer@rBS:~/ohsu/562/homework_5$
```