UNIVERSITÄT WIEN
CSLEARN - EDUCATIONAL TECHNOLOGIES

Natural Language Processing

Exercise Sheet 8

Analyzing Sentence Structure

Exercise 1

Write a recursive function to traverse a tree and return the depth of the tree, such that a tree with a single node would have depth zero. (Hint: the depth of a subtree is the maximum depth of its children, plus one.) Test your function with the two trees produced by the `ChartParser` for the `groucho_grammar` and the sentence "I shot an elephant in my pajamas" from this chapter. The result can be verified with the `Tree.height()` function.

Exercise 2

Write a recursive predicate in SWI-Prolog with the same functionality as in Exercise 1 and test it the same way.

Exercise 3

Write a recursive function `bracketing(tree)` that produces a nested bracketing for a `tree`, leaving out the leaf nodes, and displaying the non-terminal labels after their subtrees. Consecutive categories should be separated by space. Test your function with the tree:

```
from nltk.corpus import treebank
t = treebank.parsed_sents('wsj_0001.mrg')[0]
print(t)
(S
  (NP-SBJ
    (NP (NNP Pierre) (NNP Vinken))
    (, ,)
    (ADJP (NP (CD 61) (NNS years)) (JJ old))
    (, ,))
  (VP
    (MD will)
    (VP
      (VB join)
      (NP (DT the) (NN board))
      (PP-CLR
        (IN as)
        (NP (DT a) (JJ nonexecutive) (NN director)))
      (NP-TMP (NNP Nov.) (CD 29))))
  (. .))
```

```
bracketing(t)

[[[NNP NNP]NP , [[CD NNS]NP JJ]ADJP ,]NP-SBJ [MD [VB [DT NN]NP
[IN [DT JJ NN]NP]PP-CLR [NNP CD]NP-TMP]VP]VP .]S
```

Exercise 4

Write a Definite Clause Grammar in SWI-Prolog for the context-free grammar `grammar1` in Figure 3.1 and use it to parse the sentence "the dog saw a man in the park". The program should produce the following output:

```
s
    np
        det / the
        n / dog
    vp
        v / saw
        np
            det / a
            n / man
            pp
                p / in
                np
                    det / the
                    n / park
s
    np
        det / the
        n / dog
    vp
        v / saw
        np
            det / a
            n / man
        pp
            p / in
            np
                det / the
                n / park
```

Exercise 5

Write a Definite Clause Grammar in SWI-Prolog for the recursive context-free grammar `grammar2` in Figure 3.3 and use it to parse the sentences "the angry bear chased the frightened squirrel" and "Chatterer said Buster thought the tree was tall". The program should produce the following output:

```
s
    np
        det / the
        nom
            adj / angry
            nom
                n / bear
    vp
        v / chased
        np
            det / the
            nom
                adj / frightened
                nom
                    n / squirrel

  s
    np
        propn / Chatterer
    vp
        v / said
        s
            np
                propn / Buster
            vp
                v / thought
                s
                    np
                        det / the
                        nom
                            n / tree
                    vp
                        v / was
                        adj / tall
```

Exercise 6

Modify the functions `init_wfst()` and `complete_wfst()` from Figure 4.4 in this chapter so that the contents of each cell in the WFST is a set of non-terminal symbols rather than a single non-terminal. Test your function with the `groucho_grammar` and the sentence "I shot an elephant in my pajamas".

Change the line:

```
NP -> Det N | Det N PP | 'I'
```

to:

```
NP -> Det N | NP PP | 'I'
```

to verify in the trace of `complete_wfst()` that there are now two lines for `cell(1,7)`:

```
[1]    V [2]   NP [7] ==> [1]   VP [7]
[1]   VP [4]   PP [7] ==> [1]   VP [7]
```

Change the line:

```
VP -> V NP | VP PP
```

to:

```
VP -> V NP
VPC -> VP PP
```

and check that `cell(1,7)` now contains `{VPC, VP}`.
Finally, change the line:

```
S -> NP VP
```

to:

```
S -> NP VP | NP VPC
```

and check that now there are two lines in the trace of `complete_wfst()` for the `cell(0,7)`:

```
[0]   NP [1] VPC [7] ==> [0]    S [7]
[0]   NP [1]  VP [7] ==> [0]    S [7]
```

Exercise 7

Modify the function `complete_wfst()` so that when a non-terminal symbol is added to a cell in the WFST, the content of the variable `mid` is also added, i.e. we add a tuple `(symbol, mid)`. In `init_wfst()`, use `(symbol, i+1)` instead. Change also the function `display()` accordingly. Test your implementation with the final grammar from Exercise 6 and the sentence "I shot an elephant in my pajamas". It should produce the following output:

```
WFST     1          2          3          4          5          6          7
0        {(NP, 1)}  .          .          {(S, 1)}   .          .          {(S, 1)}
1        .          {(V, 2)}   .          {(VP, 2)}  .          .          {(VP, 2), (VPC, 4)}
2        .          .          {(Det, 3)} {(NP, 3)}  .          .          {(NP, 4)}
3        .          .          .          {(N, 4)}   .          .          .
4        .          .          .          .          {(P, 5)}   .          {(PP, 5)}
5        .          .          .          .          .          {(Det, 6)} {(NP, 6)}
6        .          .          .          .          .          .          {(N, 7)}
```

Exercise 8

Use the extended WFST from Exercise 7 to retrace the parse trees for our example sentence “I shot an elephant in my pajamas”. Write a recursive function `retrace(WFST, tokens)` (the second parameter `tokens` contains the token list ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas'] for our example sentence). Start with `cell(0,7)` (or `cell(0,len(tokens))` in general) and use the information in `mid` to follow the productions to `cell(0,mid)` and `cell(mid,7)`, and so on. If we reach a terminal symbol, i.e. a `cell(i,i+1)`, the corresponding token from `tokens` shall be displayed. The function should produce the following output:

```
S    -> NP   -> I
        VPC  -> VP   -> V    -> shot
                        NP   -> Det  -> an
                                N    -> elephant
                PP   -> P    -> in
                        NP   -> Det  -> my
                                N    -> pajamas
        VP   -> V    -> shot
                NP   -> NP   -> Det  -> an
                                N    -> elephant
                        PP   -> P    -> in
                                NP   -> Det  -> my
                                        N    -> pajamas
```

Exercise 9

Process each tree of the Penn Treebank Corpus sample `nltk.corpus.treebank` and extract the productions with the help of `Tree.productions()`. Discard the productions that occur only once and those that are lexical (i.e. the right-hand side contains at least one terminal token). Productions with the same left-hand side can be collapsed using a dictionary with the left-hand sides as keys and sets of right-hand sides as values.

Print the value for the left-hand side 'NP' using the format: `DT JJS NN NN | DT VBG NN NN | DT NNP CD NN | DT NN NNS ...`