



## Assignment 09: Trees and X-bar theory

handed out: Monday 22<sup>nd</sup> July, 14:00  
to be submitted by: Monday 29<sup>th</sup> July, 14:00

### Task 1: Traversal [2 points]

Implement a pre-order and a post-order traversal by filling in the bodies of the methods `Tree.preorder(self)` and `Tree.postorder(self)`. The methods should take a tree node as an argument and return a list of the nodes of the tree in the respective order (careful: not a list of the *names* of the nodes, but a list of the node objects themselves, so you can later perform operations on the subtrees in the list. You can use the given function `names(nodelist)` to turn a list of tree nodes into a list of just the names of these nodes). You can take the existing implementation of `inorder(self)` as a reference, and use the given printout methods to have your tree objects nicely displayed during testing.

*Examples:*

```
#####
# Defining a book tree object
#####
# - Book
# - - Chapter 1
# - - - Section 1.1
# - - - Section 1.2
# - - Chapter 2
# - - - Section 2.1
# - - - - Subsection 2.1.1
# - - - - Subsection 2.1.2
# - - Chapter 3
> book = Tree('Book', [
    Tree('Chapter 1', [Tree('Section 1.1'), Tree('Section 1.2')]),
    Tree('Chapter 2', [Tree('Section 2.1',
        [Tree('Subsection 2.1.1'), Tree('Subsection 2.1.2')])]),
    Tree('Chapter 3')
])

#####
# Printing out the tree object
#####

> book.printout_online()
[Book [Chapter 1 [Section 1.1], [Section 1.2]], [Chapter 2 [Section 2.1
[Subsection 2.1.1], [Subsection 2.1.2]]], [Chapter 3]]

> book.prinout_plain()
Book
```

```

Chapter 1
    Section 1.1
    Section 1.2
Chapter 2
    Section 2.1
        Subsection 2.1.1
        Subsection 2.1.2
Chapter 3

> book.printout_fancy()
|--Book
|   |--Chapter 1
|   |   |--Section 1.1
|   |   |--Section 1.2
|   |
|   |--Chapter 2
|   |   |--Section 2.1
|   |       |--Subsection 2.1.1
|   |       |--Subsection 2.1.2
|   |
|   |--Chapter 3
|
#####
# Preorder and postorder
#####
> Tree.names(book.preorder())
['Book', 'Chapter 1', 'Section 1.1', 'Section 1.2', 'Chapter 2',
'Section 2.1', 'Subsection 2.1.1', 'Subsection 2.1.2', 'Chapter 3']
> Tree.names(book.postorder())
['Section 1.1', 'Section 1.2', 'Chapter 1', 'Subsection 2.1.1',
'Subsection 2.1.2', 'Section 2.1', 'Chapter 2', 'Chapter 3', 'Book']

```

## Task 2: X-bar check [6 points]

Before approaching this task, please read up on the theoretical background in `theory-ex.09.pdf`.

Write method `Tree.is_xbar(self)` that, given a tree object as an input, returns `True` or `False` depending on whether or not the tree is in accordance with the rules of X-bar theory. To do so, fill in the missing bodies in the sections commented "check projection levels" (you can take the given implementation of `Tree.is_phrase_projection(self)` as an example on how to write the other methods in the block) and "check structural constraints" (you can take the given implementation of `Tree.is_complement_structure(self)` as an example on how to write the other methods in the block).

*Examples:*

```

#####
# Definitions of some tree objects to play around with
#####
> cat_terminal = Tree('cat')
> cat_N = Tree('N0', [cat_terminal])
> cat_Nbar = Tree('N1', [cat_N])
> cat_NP = Tree('N2', [cat_Nbar])

```

```

> the = Tree('D2', [Tree('D1', [Tree('D0', [Tree('the')])])])
> the_cat = Tree('N2', [the, cat_Nbar])
> small = Tree('A2', [Tree('A1', [Tree('A0', [Tree('small')])])])
> small_cat = Tree('N1', [small, cat_Nbar])
> on_the_mat = Tree('P2', [Tree('P1',
    [Tree('P0', [Tree('on')])],
    Tree('N2',
        [Tree('D2', [Tree('D1', [Tree('D0', [Tree('the')])])]),
        Tree('N1', [Tree('N0', [Tree('mat')])])])])
    ])]

> cat_wrong_category_mismatch = Tree('P2', [the, Tree('N1', [cat_N])])
> cat_wrong_two_heads = Tree('N1', [cat_N, cat_N])

#####
# Some tests for projection levels
#####
> the_cat.is_xbar()
True
> small_cat.is_xbar()
True
> cat_wrong_category_mismatch.is_xbar()
False # because categories of phrase and bar don't match,
# so it's not a proper phrase, so not a proper x-bar structure

> cat_N.is_head_projection()
True
> cat_Nbar.is_bar_projection()
True
> cat_NP.is_phrase_projection()
True
> cat_NP.is_bar_projection()
False # because it is a phrase, not a bar projection
> cat_wrong_category_mismatch.is_xbar()
False # see above

#####
# Some tests for structural constraints
#####
> the_cat.is_no_specifier_structure()
False # because it doesn't branch unary into just bar,
# but binary with the specifier "the"
> the_cat.is_specifier_structure()
True # same argument

> small_cat.is_no_adjunct_or_complement_structure()
False # because it has the adjunct "small"
> small_cat.is_adjunct_structure()
True # because it has the adjunct "small"
> small_cat.is_complement_structure()
False # because there is no complement

```

```

> on_the_mat.is_complement_structure()
True # because it has the complement "the mat"

> cat_N.is_head_structure()
True
> cat_NP.is_head_structure()
False
> cat_wrong_to_heads.is_head_structure()
False # because it doesn't branch unary into 1, but binary into 2 heads

```

**Important:** One of the main purposes of this assignment is to practice *recursion*. The well-formedness of a tree is a recursive concept, so it needs to be recursively verified. For example, in the second illustration of an ill-formed tree in the theory file, the ill-formedness is hidden inside a substructure: There is a category mismatch between DP and P', so the DP specifier is not well-formed, *so the NP as a whole isn't either*. Although on the top-most level, the labels make it look like an NP branching into a specifier and a bar projection is in accordance with the rules of X-bar theory, the internal structure of the DP is flawed, and therefore the surrounding NP tree, which has the ill-formed DP as a subtree, does not qualify as a well-formed X-bar tree in whole. Your program must detect that. In the methods you write, you have to carry out recursive function calls on the tree and descend into the substructure of a tree. It does not suffice to just check the name of the labels on the top-most levels, because the labels might be lying; if your tree verification ends at the first level of children of the root node, it is wrong. W.r.t. the example given here, it does not suffice to check that the node labelled NP branches into something labelled DP and something labelled N'; instead, you have to check that the DP is indeed a well-formed phrase projection and that the N' is an actual bar projection by making recursive function calls on the child nodes of the NP. Verify the behavior of your functions using the tests provided, and try to understand why some of the structures are not in accordance with the rules of X-bar theory while others are.

To help you understand the way the functions interact with each other and where the recursion enters, here is an illustration of the call hierarchy:

