Assignment 7

Ethan Poole
LING 185A: Comp. Ling. I
Due: 26 May 2019

Instructions: Download `CFG.hs` and `Assignment07.hs` from the course website into the same directory on your computer. The **`import`** line near the top of `Assignment07.hs` imports all of the definitions from `CFG.hs`, which you may then use in your assignment. You should familiarize yourself with everything in `CFG.hs` before beginning the assignment. Please submit your assignment as one file: a modified version of `Assignment07.hs`.

Note: The type signatures are only given in the stub file, as they are quite long.

1 *4 points*

Please write the following functions for practice with some of the relevant types:

(1) `terminalsOnly`: tries to convert a mixed string of nonterminal and terminal symbols into a string of just terminal symbols. If the given string contains no nonterminal symbols, then the result should be a **`Just`** value with the list of whatever terminal symbols the given string contains. If the given string contains nonterminal symbols, then the result should be **`Nothing`**.

Example usage:

```
terminalsOnly [T "John", T "saw", T "Mary"]
                    ⟹* Just ["John", "saw", "Mary"]
terminalsOnly [T "John", T "saw", NT "NP"] ⟹* Nothing
```

(2) `yields`: returns the list of terminal symbols that appear at the leaves of the given tree, in order.

Example usage:

```
yield (NonLeaf "S" (Leaf "NP" "Mary") (Leaf "VP" "ran"))
                    ⟹* ["Mary", "ran"]
yield (NonLeaf 0 (Leaf 0 'a') (NonLeaf 0 (Leaf 0 'b') (Leaf 0 'c')))
                    ⟹* "abc"
```

$\boxed{\quad\quad\quad 2}$ *15 points*

Please write the following functions that convert between trees, rule lists, and derivations:

(3) `treeToRuleList`: returns the list of rules used in the leftmost derivation corresponding to the given tree (in the order in which they are used in that leftmost derivation).

Hint: All the rewriting steps that construct parts of a node's left daughter tree precede all the rewriting steps that construct parts of that node's right daughter tree . . .

Example usage:

`treeToRuleList (NonLeaf "S" (Leaf "NP" "Mary") (Leaf "VP" "ran"))` $\Longrightarrow^*$
`[NonterminalRule "S" ("NP", "VP"), TerminalRule "NP" "Mary", TerminalRule "VP" "ran"]`

`treeToRuleList (NonLeaf 1 (NonLeaf 2 (Leaf 4 'a') (Leaf 5 'b')) (Leaf 3 'c'))` $\Longrightarrow^*$
`[NonterminalRule 1 (2,3), NonterminalRule 2 (4,5), TerminalRule 4 'a', TerminalRule 5 'b', TerminalRule 3 'c']`

`treeToRuleList (NonLeaf 1 (Leaf 2 'a') (NonLeaf 3 (Leaf 4 'b') (Leaf 5 'c')))` $\Longrightarrow^*$
`[NonterminalRule 1 (2,3), TerminalRule 2 'a', NonterminalRule 3 (4,5), TerminalRule 4 'b', TerminalRule 5 'c']`

(4) `ruleListToTree`: in effect, the reverse of `treeToRuleList`. If the given list of rules does not correspond to any leftmost derivation, then the result should be **`Nothing`**. If the given list of rules does correspond to a leftmost derivation, the result should be a **`Just`** value containing the appropriate tree.

By "leftmost derivation" here, I mean a sequence of steps from a single nonterminal symbol to a string containing only terminal symbols. As this is not relative to any particular grammar, it does not matter what nonterminal symbol the derivation starts from; this will just be whatever nonterminal symbol happens to appear on the left-hand side of the first rule in the list.

Example usage:

```
ruleListToTree [NonterminalRule "S" ("NP", "VP"), TerminalRule "NP" "Mary", TerminalRule "VP" "ran"]
```
$\Longrightarrow^*$ `Just (NonLeaf "S" (Leaf "NP" "Mary") (Leaf "VP" "ran"))`

```
ruleListToTree [NonterminalRule 1 (2,3), TerminalRule 2 'a', NonterminalRule 3 (4,5), TerminalRule 4 'b']
```
$\Longrightarrow^*$ `Nothing`

(5) `treeToDerivation`: returns the leftmost derivation corresponding to the given tree, in the form of a list of mixed strings containing terminal and nonterminal symbols.

You must not execute this function by first recovering the list of rules used in the derivation and then carrying out the rewriting steps one by one. There is a much more elegant and clever way to construct the necessary strings *directly* from the tree structure.

Example usage:

```
treeToDerivation (NonLeaf "S" (Leaf "NP" "Mary") (Leaf "VP" "ran"))
```
$\Longrightarrow^*$ `[[NT "S"], [NT "NP", NT "VP"], [T "Mary", NT "VP"], [T "Mary", T "ran"]]`

```
treeToDerivation (NonLeaf 1 (NonLeaf 2 (Leaf 4 'a') (Leaf 5 'b')) (Leaf 3 'c'))
```
$\Longrightarrow^*$ `[[NT 1],[NT 2,NT 3],[NT 4,NT 5,NT 3],[T 'a',NT 5,NT 3],[T 'a',T 'b',NT 3],[T 'a',T 'b',T 'c']]`

3

10 points

The `rewrite` and `splitAtNT` functions provided in `CFG.hs`, which we discussed in class, capture the basic idea of what a rewriting step looks like. However, that particular rewrite function does not necessarily carry out the kinds of steps that appear in leftmost derivations. It rewrites, fairly arbitrarily, the leftmost occurrence (if there is one) of the particular nonterminal symbol to which the given rule can apply. This will not in general be the leftmost nonterminal symbol in the string.

Your task here is to write the two following functions, working towards carrying out the leftmost derivations allowed by a grammar.

(6) `splitAtLeftmost`: splits a list of symbols at the leftmost occurrence of a nonterminal symbol. The idea is to use this function to pick out the position where the next rewriting step will happen in a leftmost derivation. If there

are no nonterminal symbols in the list, then the result should be **`Nothing`**. Otherwise, it should be a **`Just`** value containing a triple, which has the leftmost nonterminal symbol itself as its second component, and has the preceding and following parts of the list as the first and third components respectively.

Example usage:

```
splitAtLeftmost [T "apples", T "and", NT "NP", T "or", NT "NP"]
        ⟹* Just ([T "apples", T "and"], "NP", [T "or", NT "NP"])
```

```
splitAtLeftmost [T "apples", T "and", T "oranges", T "or", T "bananas"]
                                                          ⟹* Nothing
```

(7) `rewriteLeftmost`: rewrites the leftmost nonterminal symbol in the given list of symbols, in all the ways possible according to the given list of rules. Each such possible rewriting step produces a new list of symbols. The result should be a list containing all these resulting *lists of symbols*. In the following examples, **`snd`** is used to extract the second component of a pair, here, for the purposes of retrieving a CFG's list of rules.

Example usage:

```
rewriteLeftmost (snd cfg1) [NT "NP", NT "VP"] ⟹*
[[NT "D", NT "N", NT "VP"], [T "John", NT "VP"], [T "Mary", NT "VP"]]
```

```
rewriteLeftmost (snd cfg1) [T "John", NT "VP"]
                                   ⟹* [[T "John", NT "V", NT "NP"]]
```