CENG 242

Hw #3

Spring 2007/2008

(Due: April 13th, 2008 Sunday 23:59)

In this homework, you will implement a simplified CFG (context free grammar). In this simplified CFG, there will be non-terminal symbols represented by upper case English characters (‘A’ to ‘Z’) and terminal symbols represented by lower case English characters (‘a’ to ‘z’), and the rules are defined as *S* -> *w* where *S* is a single non-terminal symbol and *w* is string of terminals/non-terminals or empty symbol. Empty symbol is represented by the character ‘#’. Also this simplified CFG will not have any type of recursion. An example is:

S -> aTb
S -> U
U -> Tbc
T -> cc
T -> #
T -> aa

For the definition and details of CFG, you can have a look at
http://en.wikipedia.org/wiki/Context-free_grammar

You should have a module called CFG having the following functions (to clarify what exactly they do, look at the example section):

- `createCFG :: Char -> CFG`

This function will create a CFG getting a start variable. The internal representation of the CFG is up to you.

- `insertRule :: CFG -> (Char,String) -> CFG`

This function will get a CFG and add the given rule to CFG and return it.

- `deleteRule :: CFG -> (Char,String) -> CFG`

This function will delete the given rule from given CFG and return the resulting CFG.

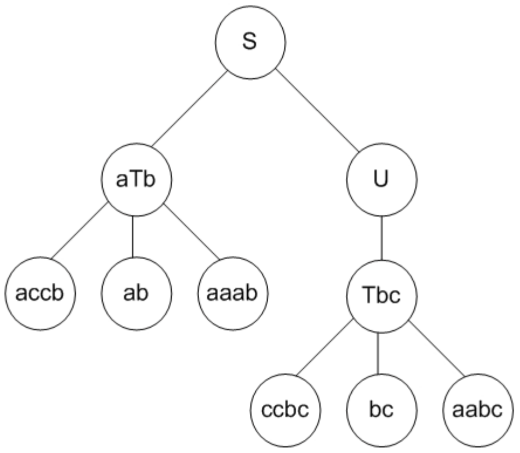
- `canGenerate :: CFG -> String -> Int`

This function will return in how many ways, the given string can be generated by the given CFG. All the characters in the given string will be terminals.

- `show :: CFG -> String`

This function will be the instance of Show that will show the grammar in tree format. Detail of the tree format is given below.

For the CFG given above, the tree will be:



and it should be shown as:

S aTb:S|U:S accb:aTb:S|ab:aTb:S|aaab:aTb:S|Tbc:U:S ccbc:Tbc:U:S|bc:Tbc:U:S|aabc:Tbc:U:S	The start symbol Level 2 nodes represented with their parents Level 3 nodes represented with their parents Level 4 nodes represented with their parents The strings will be '|' separated, no blanks in between any characters, each line is newline character separated. The order of lines is important but order in the same line is not important.

Example:

`c1 = createCFG 'S'`	`create the CFG with start symbol S`
`c2 = insertRule c1 ('T',"aUb")`	`insert the rule T -> aUb`
`s1 = show c2`	`the result is shown below`
`c4 = insertRule c2 ('T',"aaVa")`	`insert the rule T -> aaVa`
`c5 = insertRule c4 ('V',"a")`	`insert the rule V -> a`
`c6 = insertRule c5 ('T',"V")`	`insert the rule T -> V`
`c7 = insertRule c6 ('V',"#")`	`insert the rule V -> ε`
`c8 = deleteRule c7 ('S',"a")`	`no such rule, so return the same CFG`
`c9 = deleteRule c8 ('V',"a")`	`the rule V -> a is deleted`
`c10 = insertRule c9 ('T',"UaV")`	`insert the rule T -> UaV`

`c11 = insertRule c10 (‘V’,”b”)`	`insert the rule V -> b`
`s2 = show c11`	`the result is shown below`
`i1 = canGenerate c11 “a”`	`will return 0`
`c12 = insertRule c11 (‘S’,”Va”)`	`insert the rule S -> Va`
`i2 = canGenerate c12 “ab”`	`will return 0`
`s3 = show c12`	`the result is shown below`
`c13 = insertRule c12 (‘T’,”V”)`	`the rule is present, don’t insert`
`c14 = insertRule c13 (‘T’,”ab”)`	`insert the rule T -> ab`
`c15 = insertRule c14 (‘S’,”Tb”)`	`insert the rule S -> Tb`
`c16 = insertRule c15 (‘T’,”#”)`	`insert the rule T -> ε`
`s4 = show c16`	`the result is shown below`
`i3 = canGenerate c16 b`	`will return 2`

The results of the s1, s2, s3 and s4:

s1	S (Note: It should end with newline character)
s2	S (Note: It should end with newline character)
s3	S Va:S a:Va:S|ba:Va:S
s4	S Va:S|Tb:S a:Va:S|ba:Va:S|aUbb:Tb:S|aaVab:Tb:S|Vb:Tb:S|UaVb:Tb:S|abb:Tb:S|b:Tb:S aaab:aaVab:Tb:S|aabab:aaVab:Tb:S|b:Vb:Tb:S|bb:Vb:Tb:S|Uab:UaVb:Tb:S|Uabb:UaVb:Tb:S

Explanation for i1, i2 and i3:

i1: Since ‘S’ has no definition yet, we return 0
i2: Since ‘S’ has only one definition (S->Va), it can generate only “a” and “ba”, so it can not generate “ab”, so return 0
i3: “b” can be generated either by S -> Tb -> Vb -> b or S -> Tb -> b. So we return 2

Specifications:

- All the work should be done **individually**.
- Your codes should be written in **Haskell** and have the name “**CFG.hs”.**
- Your code should have the module **CFG** and export only the given methods.
  module CFG(CFG, createCFG, insertRule, deleteRule,canGenerate,show) where
- In evaluation, black box method will be used. So, be careful about names, types etc.
- You will submit your codes through **cow** system.
- You should test your codes in **inek** machines with **hugs** before submitting.