**COMS 4701 – Artificial Intelligence – Fall 2014**

**Assignment 1: Sentence Generator in Lisp**

**(version 1, September 9, 2014)**

**Due: Wednesday, September 24, 11:59 PM**

The aim of this project is to write a program in Lisp that generates random sentences in English. For this purpose you will use Context-Free Grammars (CFG), a standard framework for describing natural language.

We are providing you with a small corpus of sentences taken from news articles in CNN. You will use this corpus to build your grammar, and produce similar sentences randomly. The corpus is located on *~cs4701/Project1/corpus.txt*, and is subject to change. Use it as a starting point.

We are giving you some code that already works, but you will have to improve it to produce more complex sentences while avoiding over-generation. The starting code can be found at *~cs4701/Project1/grammar.lisp*

The lecture notes include an introduction to CFG as well as some external pointers to useful web sites about language and syntax. Here are additional useful links:

<http://people.umass.edu/scable/LING201-SP13/Slides-Handouts/Syntax-Basics.pdf>

<http://www.linguistics.ucla.edu/people/Kracht/courses/ling20-fall07/ling-intro.pdf>

<http://en.wikipedia.org/wiki/Linguistic_competence>

**I) Context Free Grammar**

The code already has the following simple grammar:

sentence -> noun-phrase verb-phrase

noun-phrase -> Article Noun

verb-phrase -> Verb noun-phrase

Article -> the a

Noun -> man ball woman table

Verb -> hit took saw liked

In this example, we have named “pre-terminals” (parts of speech) using mixed case. The other non-terminals are in all lowercase as are the words themselves. Note that as far as Lisp is concerned, case doesn’t matter here.

1) Build a more sophisticated grammar **(60 points)**

Here are some suggestions:

* Add non-terminal symbols such as Adjective, Preposition, Conjunction, RelativeClause, etc. **(10 points)**
* Add more sophisticated production rules such as “noun-phrase -> Article Adjective Noun | Article Noun”. This example would allow sentences such as “the tall man saw the blue ball”. **(10 points)**
* Create non-terminal symbols to handle singular or plural nouns, present or past tense verbs, transitive and intransitive verbs, etc. **(10 points)**
* Based on the sentence corpus, add more terminal symbols (words) to expand the pre-terminal symbols Article, Noun, Verb, etc. **(10 points)**
* You should not put any punctuation in your grammar (the sentences in the corpus don’t have any), and you don’t need to worry about capital letters.

The grammars should be COMPETENT enough so that you can produce arbitrarily long constituents, e.g., 10 prepositional phrases or a conjunction of 10 sentences, e.g., “The man saw the cat and the cat saw the mouse and the mouse saw the man and the man saw the woman and the woman saw the red yellow green small red mouse”.

Document the grammar that you used in your report. If you add new symbols and new production rules, you should name them appropriately, and explain why you added them (give example sentences).

The function that produces the random sentences should be named random-sentence and take one argument (a non-terminal symbol).

You will also output a file named *q1\_uni.txt* containing 10,000 sentences randomly generated by your initial grammar.

Your grammar will be graded based on the symbols and production rules that you used. **(10 points)**

We will also select randomly sentences from your output file containing the 10,000 sentences produced by your function random-sentence to check their grammaticality. **(10 points)**

2) Create rules to generate specified sentences **(20 points)**

You will have to create a set of rules from your grammar that will generate the first five sentences of the corpus.

For instance, with the sentence “The man saw the table”, the tree that generates this sentence is the following:

**sentence**

/ \

**noun-phrase verb-phrase**

/ \ / \

**Article Noun Verb noun-phrase**

| | | / \

the man saw **Article Noun**

| |

a table

The rules have to be given in depth-first traversal order.

The list of rules for this example would thus be:

'((sentence 0)

(noun-phrase 0)

(Article 0)

(Noun 0)

(verb-phrase 0)

(Verb 2)

(noun-phrase 0)

(Article 1)

(Noun 3))

And when we give this set of rules to the function targeted-sentence*,* it generates the sentence “the man saw the table”.

So you have to write the five sets of rules that, using your grammar, will produce the first five sentences of the corpus.

Name rules1, … rules5 this set of rules, and define them as global parameters in your code, e.g.,

(defparameter rules1 '((sentence 0) (noun-phrase 0) (Article 0) (Noun 0) (verb-phrase 0) (Verb 2) (noun-phrase 0) (Article 1) (Noun 3))))

Include these rules and their output in your report.

**II) Rejecting sentences (20 points)**

Even if your sentences might be syntactically correct, many of them might not make much sense in English. For instance sentences like “The table hit the table” is possible, but doesn’t make much sense.

It is not difficult to reject some trivial cases. We could for instance reject sentences that have the same words twice (except for closed-class words like articles, prepositions that often appear several times in a sentence).

You should write a function that:

* rejects sentences that are too long (set a limit N1 on the maximum number of words, for instance 40) **(5 points)**
* rejects sentences with tree depth larger than N2, for instance 10 **(5 points)**
* rejects sentences where the same word appears twice (except some closed class words, e.g., prepositions, articles, conjunctions (‘the’, ‘a’, ‘and’, ‘but’…) **(5 points)**
* rejects sentences based on at least two other criteria of your choice **(5 points)**

The function will thus limit the sentences to some reasonable complexity.

The function that generates these valid sentences should be named generateValid*,* and takeone argument (a non-terminal symbol).

You can if you want have a function *validp* that returns true if a sentence is valid.

Additionally, output a file named *q2\_uni.txt* containing 100 sentences that have been rejected by your function.

**LISP hints**

To load a Lisp file, use the function

(load “your\_file”)

You can write to a file using the function *with-open-file*

(with-open-file (outfile "your\_file" :direction :output)

(prin1 '(here is an example) outfile))

**Submission**

We will send you by mail a key that you will use to create a folder that only you and us will have access to.

For that, login on the clic machines and run the script make-hidden.sh with the key that we gave you (94942234878989808 here for instance)

aa1234@beijing:~$ bash ~cs4701/make-hidden.sh 94942234878989808

This will create a hidden folder with folder whose name is your key inside.

Put all your files in this folder.

Also, to be sure that we can access it, you might have to change the permissions on your account:

When you are logged in your clic account, do:

cd

cd ..

chmod go+x uni

What to submit:

* your code in one file named *grammar\_uni.lisp*. It has to run on clisp on the clic machines.
* A report in text format named *report\_uni.txt* that explains what you did.
* The two output files containing the sentences that you generated, named *q1\_uni.txt* and *q2\_uni.txt*