Assignment 2 - Report

Yehuda Gihasi 305420671, Elron Bandel 308130038

December 18, 2019

1 Part 0 - Generating sentences from a CFG

I this part we Implemented the ability to genrate multiple sentences with the Command Line Argument Flag -n NUM.

2 Part 1 - Weights

2.1 a. Causes Of Long Sentences

Sum Rules contain recursive call for example: $NP \rightarrow NP, PP$ which can cause redundent recursive call that will produce very long sentense. if this sentence is weighted in the same manner as $NP \rightarrow Det, Noun$ the sentence will be extended with 0.5 probability.

2.2 b. Reasons For Multiple Adjective Rarity

There are 6 Rules starting with $Noun$ and only one of them ($Noun \rightarrow Adj, Noun$) producing Adjectives ($Adj$) but in order to produce multiple adjectives this rule need to be called recursivly. The Probabilty for every call is very small and the probabilty to get them twice in a row is multipication of this small probabilty which is even much smaller. Thus, getting this event is extremly rare.

2.3 c. Fixes of a. & b:

2.3.1 a.

$NP \rightarrow Det, Noun$ will be modified to be used more often. the weight we choose for this rule is 3

2.3.2 b.

$Noun \rightarrow Adj, Noun$ will be modified to be used more often. the weight we choose for this rule is 3

2.3.3 Extra

In order to make the sentences more tuned with reality we decide to give more weight to the rule: $ROOT \to S$.

3 Part 2 - Extending The Grammar

In this part we tried to generalize the grammar from the sentences. the way we did it was to analyse the unique grammar tree of every tree and derive the grammar rules from it. We had to decide in every tree what rules are rules that we have seen before and what rules represent different structures we have never seen. Other than just getting the rules we tried to generalize as much as possible and to give them weights that will represent the unique corpse but at the same time will suit the the grammar we had from the previous tasks. We left full description of our steps of generating the new rules of this part in the comments on the grammar2 file.

3.1 Modifications

In order to extract rules in the right way we did two main modifcations

Terminals We had to make a distinction between different Non Terminals Parts Of Speech, for example: between Verb and Verb-d which is verb that cant be aimed towrdes object. Or between Noun and Noun-u which is Uniqe Noun or Proper Noun that relate to specific Noun and cant have the Determinent 'a' before him for example.

Non-Terminals In some cases we identified unique structure that is show dependency between othe levels in the grammar trees. In this case we used new Notation A
B to represent A that was derived from B. or A-B that is producing A,B. this Notations helped us to deal we cases of multi-level dependency. One example of this use is: $CC - VP \to Conj, VP/CC - VP$ that helped us to "remember" in what part of genrating Conjuctive Sentence we are and which one were heading to.

3.2 Bad Interaction

the following sentences have grammtic similarities: (b) Sally and the president wanted and ate a sandwich . (h) Sally is lazy . (i) Sally is eating a sandwich. They are all starting with the shared Proper Singular Noun that in most cases will lead to Singular Verb but in on of the cases interaction with Conjuctor made the Single - Plural and created a continues grammatical issue.

4 Part 3 - Tree Structures

In this part we implemented the mechanism that generate grammar trees from given rules. we used it to debug part number 2.

5 Part 4 - Additional Lingusitic Structures

we choose to implement a/an and Yes/No question (a+b)

a: We added 2 rules $NP \rightarrow a\ Noun$, $NP \rightarrow an\ AEINoun$(for noun start with vomel letters), than I should handle with the case that there is an Adj before the noun, so I should split it for 4 rules to handle with all the permutation of vomel Adj and non vomel Adj, and vomel Noun and non vomel Noun, Noun → Adj AEINoun. Noun → Adj AEINoun, Noun → Adj Noun, AEINoun → AEIAdj AEINoun , AEINoun→ AEIAdj Noun. Than I an Adv rules before the Adj so its 3 rules (because one rule we already have from Part 2). AEIAdj → AEIAdv Adj, AEIAdj→ AEIAdv AEIAdj, and this rule: Adj→Adj AEIAdv AEIAdj is to avoid (a extraordinary for example).

b: I start to implement it from the simple question. And add the rule :ROOT → QYN Than I add the question format: QYN→ ynqs Noun-u VerbP NP ?, (yqns - will/did, VerbP to eat, drink etc) Than I expand it to all the "is . . . .?" Question with all the variations. First handle with "is noun Adj" and than "is noun a
an noun" than handle with "is noun verb noun" And so one to get all the options of yes/no question. This is an example for one of the rule that we produce: QYN → Verb-z (is) noun-u (sally) Adj (lazy). We had a problem when we create a new rules we got a permutations that was wrong so we fixed the problem using the print tree function to see which rules made the problem and than change the rules to fix it.

6 Part 5 - Extra

In order to extend our grammar we looked for something that will be used in day to day language in the one hand but in the other hand will be somwhat complex structure. We analysed structures of jokes and find one joke that was very intresting structure wise - Not Jokes. The Not Joke defined by the Urabn Dictionary "use of "...Not!" after apparently affirmative sentences in order to state the contrary" definition that lead us to more generelized idea: Post Sentence Modifier.

PSM - Post Sentence Modifier Post Sentence Modifier is a structure of framing sentence after being said in a completely different way. the general structure is a sentence that seem to be independent and than straight after modifying phrase that change the original meaning of the sentence.

Implementation And Conclusions In order to Implement PSM we implemented this structure as drivation of the ROOT. than we Implemented few Modifiers. In order to toss things around we changed the nouns to fit the new era and got few funny results.