Angela Yang

12/6/23

LING 185A

Natural Language is Not Context-Free: A Comparison Between Linear-Indexed Grammars (LIGs) and Tree-Adjoining Grammars (TAGs)

Introduction

We observe that human language accepts an infinite set of strings as grammatical and furthermore, meaningful. However, seeing as human minds are definitely finite as being contained within a human brain and acquired by babies without directly teaching all possible grammatical sentences, it is impossible for there to be simply memorization taking place. Instead, human minds must utilize some computation. Formally, in the field of mathematical and logical computation, there is a contained hierarchy of languages which can be used to accept strings called Chomsky's hierarchy. Chomsky's hierarchy composes of four levels of formal grammars: regular languages, context-free grammars, context-sensitive grammars, and turing machines with each grammar being more computationally powerful than those previous. Shieber's studies on Swiss German's non-adjacent NP V pair dependencies[1] show in-depth evidence against context-free grammars being sufficiently able to represent natural language. However, natural languages are also not quite as capable as context-sensitive grammars or turing machines, both of which are too powerful that they offer little restriction and thus, offer little

[1] Shieber, S. (1985). "Evidence against the context-freeness of natural language." Linguistics and Philosophy 8: 336.

explanation to how human computation of natural language operates[2]. In order to offer a useful and accurate computational classification of natural language, we make an intermediate class of languages termed mildly context-sensitive languages, which restrict context-sensitive abilities but can still account for the non-context-free aspect. Two such systems proposed are Linear-Indexed Grammars and Tree-Adjoining Grammars. In this essay, I will explore the formalisms of both, analyze differences in generation, and ultimately show the underlying equivalence of the two languages.

Body

First, I will introduce the formalisms, rules, and generated types of both grammars. LIGs contain three different kinds of rules: i. The stack is copied to all non-terminal daughters of a node, ii. The stack is pushed onto and passed down to one unique non-terminal daughter, iii. The stack is popped and passed down to all non-terminal daughters[3]. Other details of LIGs include that nonterminals cannot bear stacks, and that the stacks aren't bound to size[4]. LIGs generate strings this way through the expansion of the stack, similarly to how CFGs do, but with the extra nuance of a restricted stack mechanism. TAGs begin with an initial tree and have two modes of tree modification: substitution, where a tree with a root node of X type can be substituted into a tree with a leaf of X type, and adjoining, where a tree of X type root and X type leaf can be spliced into the middle of the tree where there is an X type node[5]. Since TAGs work with modifying an initial tree until you get a product, the grammar generates a tree[6].

[2] Frank, R. (2003). "Restricting grammatical complexity." Cognitive Science 28: 675.
[3] Gazdar, G. (1988). "Applicability of indexed grammars to natural languages." U. Reyle and C.: 71
[4] Gazdar, G. (1988). "Applicability of indexed grammars to natural languages." U. Reyle and C.: 71
[5] Frank, R. (2003). "Restricting grammatical complexity." Cognitive Science 28: 677.
[6] Frank, R. (2003). "Restricting grammatical complexity." Cognitive Science 28: 678.

Next, we will analyze the step-by-step derivations of both grammars using the example context-free stringset $L_1 = \{a^n b^n \mid n \geq 0\}$. A derivation of this using a CFG for reference would look like the following, with this ruleset:

1) $S \rightarrow aSb$
   $S \rightarrow \epsilon$

Here we can see the derivation of how (1) generates the string 'aaaabbbb' and how it continues for others in L_1 with any $n \geq 0$.

2) S
   aSb
   aaSbb
   aaaSbbb
   aaaaSbbbb
   aaaabbbb

With a Linear-Indexed Grammar, we can generate L_1 with the following grammar. This is written in line with Gazdar's notational rules, such that nonterminals are represented with capital letters, terminals with lowercase, as well as that [..] denotes a stack, [i,..] denotes a stack with i on top, and [] denotes the empty stack[7]:

3) $S[..] \rightarrow aA[z,..]$
   $A[..] \rightarrow aA[a,..]$
   $A[..] \rightarrow B[..]$
   $B[a,..] \rightarrow bB[..]$
   $B[z,..] \rightarrow b$

Here we can see the derivation of how (3) generates the string 'aaaabbbb' and how it continues for others in L_1 with any $n \geq 0$.

4) S[]
   aA[z]

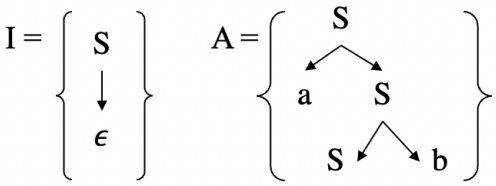
[7] Gazdar, G. (1988). "Applicability of indexed grammars to natural languages." U. Reyle and C.: 71

aaA[a,z]
aaaA[a,a,z]
aaaaA[a,a,a,z]
aaaabB[a,a,z]
aaaabbB[a,z]
aaaabbbB[z]
aaaabbbb

As an aside on analogous structures in the above examples, LIGs themselves can be considered CFGs with the addition of a stack. Consider the similarities between (2) and (4) in which instead of a and b being generated simultaneously like in (2), we store an a on the stack for each a generated and the number of b's generated is determined by the popped a's in (4).

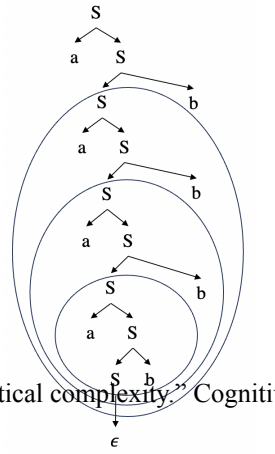
With a Tree-Adjoining Grammar, we can generate L_1 with the following grammar. The ruleset is composed of the two modifications which is characteristic of TAGs: substitution and adjoining. This is written in line with Frank's notational rules, such that the set of I represents the initial tree and A represents the auxiliary trees[8]:

5)



Here we can see the derivation of how (5) generates the string 'aaaabbbb' and how it continues for others in L_1 with any $n \geq 0$.

6)

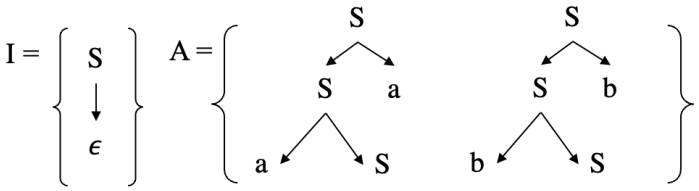
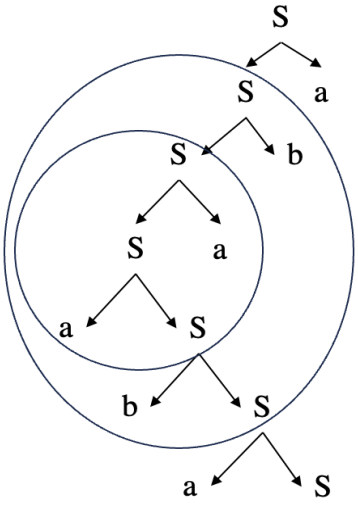


[8] Frank, R. (2003). "Restricting grammatical complexity." Cognitive Science 28: 681

LIGs and TAGs can also generate languages that aren't generable by CFGs, which makes them more powerful and gives the class of grammars the "mildly-sensitive" aspect. The difference is that CFGs are unable to generate languages where there are dependencies between non-adjacent elements, whereas LIGs and TAGs can. For example, the stringset {ww | w ∈ {a,b}*} is not generable because the first element of the second w must equal the first element of the first w, and these elements are not always adjacent, so there isn't "memory" that is able to restrict these to each other when using a CFG. LIGs and TAGs however, have memory in the form of a stack (for LIGs) and a nested embedded push-down automaton (EPDA) for TAGs which allows for nested structures to retain positions relative to each other through auxiliary trees and adjoining[9]; both "memories" can generate context sensitive grammars to a limit. The following three languages are utilized as examples:

[9] Frank, R. (2003). "Restricting grammatical complexity." Cognitive Science 28: 675

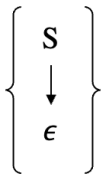
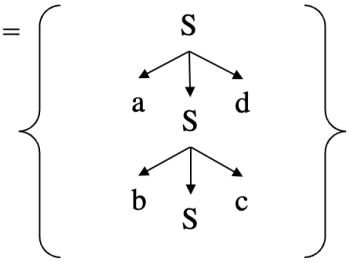
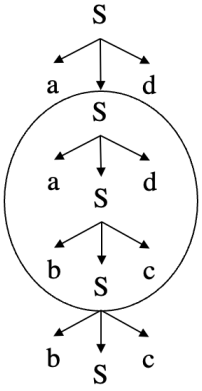
$L_2 = \{ww \mid w \in \{a,b\}^*\}$

Linear-Indexed	Tree-Adjoining
Grammar: F[..] → aF[a..] F[..] → bF[b,..] F[..] → B[..] B[a,..] → B[..]a B[b,..] → B[..]b B[] → ϵ	***Initial and Auxiliary Trees:*** 
Derivation: (w = 'aba') F[] aF[a] abF[a,b] abaF[a,b,a] abaB[a,b,a] abaB[b,a]a abaB[a]ba abaB[]aba abaaba	***Derivation:*** (w = 'aba') 

A table of derivations through each method is shown above.

We explained how L_2 isn't generable by CFG in the paragraph prior.

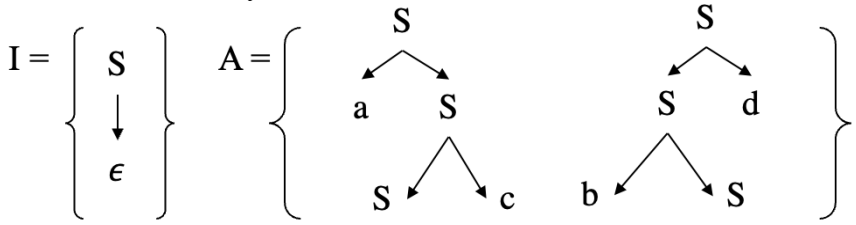
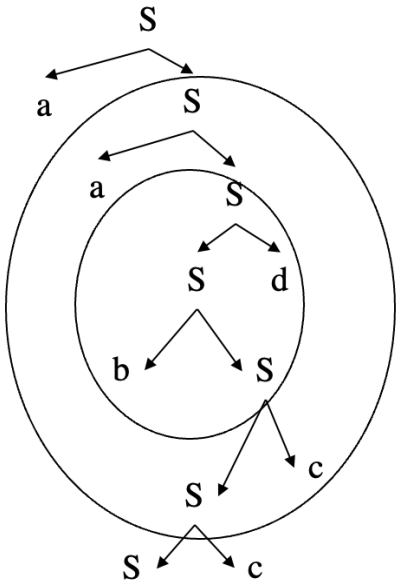
$L_3 = \{a^n b^n c^n d^n \mid n \geq 0\}$

Linear-Indexed	Tree-Adjoining
Grammar: S[..] → ϵ S[..] → aA[z,..]d A[..] → aA[a,..]d A[..] → B[..] B[a,..] → bB[..]c B[z,..] → bc	***Initial and Auxiliary Trees:*** I =  A = 
Derivation: (‘aabbccdd’) S[] aA[z]d aaA[a,z]dd aaB[a,z]dd aabB[z]cdd aabbccdd	***Derivation:*** (‘aabbccdd’) 

A table of derivations through each method is shown above.

L_3 isn’t generable by a CFG because all four sections must be equal and thus must be generated at the same time in a rule, which is impossible without “splitting” the nonterminal, since there isn’t any memory to remember how many a’s there are in order to generate nonadjacent sections such as c and d.

$L_4 = \{a^i b^j c^i d^j \mid i \geq 0, j \geq 0\}$

Linear-Indexed Grammars	Tree-Adjoining Grammars
Grammar: S[..] → ϵ S[..] → A[..] A[..] → aA[a,..] A[..] → A[d,..]d A[..] → B[..] B[a,..] → B[..]c B[d,..] → bB[..] B[] → ϵ	***Initial and Auxiliary Trees:*** 
Derivation: (‘aabccd’) S[] A[] aA[a] aaA[a,a] aaA[d,a,a]d aaB[d,a,a]d aabB[a,a]d aabB[a]cd aabB[]ccd aabccd	***Derivation:*** (‘aabbccdd’) 

A table of derivations through each method is shown above.

L_4 isn’t generable by a CFG because the number of a’s and c’s are dependent on each other but unable to be generated simultaneously as they are not adjacent to each other. If they were to be generated at once, the mass of c’s would prevent b’s and d’s to be generated adjacently. Thus, L_4 isn’t generable by a CFG.

These grammars are valuable to computational linguistics because they are more restrictive than context-sensitive grammars. I will use the motivating examples of $L_5 = \{www \mid w \in \{a, b\}^*\}$ and $L_6 = \{a^n b^n c^n d^n e^n \mid n \geq 0\}$—which can be represented with context-sensitive grammars—to show these limitations. In layman's terms, the "generative adjacency" property that I've shown CFGs to require still limits mildly context-sensitive grammars, though, in a different way. For Linear-Indexed Grammars, the stacks are able to generate things that are not immediately adjacent. However, the popping requirement of the stack means that it must be consumed to be used, whether it's storing a specific order like in L_2 or storing a number of terminals like in L_3. This destructive memory read limits mildly context-sensitive grammars. We can see this limitation in L_5, where after the generation of the second w, the stack is now empty, so it cannot be used to generate the third w. In L_6, it's a similar issue, where we can extend the writing to stack, reading from stack even property to mean it can only generate four out of the five terminal strings. A distinguishing language I'd like to point out however is that $L_7 = \{a^n b^n c^n \mid n \geq 0\}$ *would* be generable by LIGs as the a's and c's can be generated at the same time to create the stack and the b's can be generated by consuming the stack. Analogously in Tree-Adjoining Grammars, adjoining auxiliary trees that must be constrained to the structure and nodes of the main tree forces a similar effect. Parts within the tree are able to be expanded where others might not have the same ability to expand. This rewrite system is characterized by the Embedded Push-Down Automaton, which is described by Frank as a collection of stacks where the tops of any of the stacks are accessible. In this way, a language such as L_3 can preserve the positions at which a's, b's, c's, and d's may be added in the proper areas because of the structures of the adjoining trees forcing a's and b's to the left of the tree and c's and d's to the right. This structure

similarly can't generate either L_5 or L_6 as it isn't able to maintain the tops of 5 distinct stacks that must be generated simultaneously in the correct positions, or maintain the tops of 3 stacks that must generate the same things at the same time. This is because of the limits of positioning that a tree is able to record.

Discussion

To summarize, I will now go over the key similarities and differences between Linear-Indexed Grammars and Tree-Adjoining Grammars; under the formalisms however, I will also show why they generate the exact same class of string languages. In a nutshell, the generativity of mildly-context sensitive grammars is palindromic. I use this description not because what's generated must be a palindrome, but rather because the method of generation is in a palindromic style, where pieces dependent on each other must be generated at the same time, somewhat like a palindrome. In LIGs, the generation of dependent elements can sometimes be not immediate with the function of a stack, such as in the generation of a's and d's first, then b's and c's after in L_3. In TAGs, all elements dependent on one another are generated at once, with the positions preserved through the tree structure. Because of this requirement for simultaneous generation, LIGs and TAGs generate the same things: what would be in the stack for an LIG would be a part of the auxiliary tree in a TAG.

Alluding to the previous comparison about how LIGs can be created with a CFG with an additional stack, this adds an extra layer of computational ability. TAGs are also likewise able to

be created by adding the ability to grow strings “in the middle” to a CFG. We can allow for strings beginning and ending with the same nonterminal to be inserted into the CFG to simulate the same effect as tree-adjoining. By adding something like this or a stack results in the same effect: extended simultaneous generation of non-adjacent elements. This shows why LIGs and TAGs generate the same class of languages as they can be extended from CFGs with analogous methods.

LIGs and TAGs offer a compelling case for how human language processing works, and proposes that—as the initial observation states—the physically finite human brain results instead in a limit on context sensitivity rather than a limit on string generation. Further studies to confirm or flesh out the true applicability of LIGs and TAGs could include exploring where the exact limit of context sensitivity might be in different languages and the range that it may differ by. Because only some natural languages have been proven to be non-context-free, perhaps this isn’t applicable to all languages and some may be satisfied with context-free computation. Perhaps, there is more context sensitivity developing as more human language is becoming written rather than spoken through the digital age. This might allow for a higher context sensitivity because written languages allow for a bigger visual-aided memory than spoken. Computationally, LIGs and TAGs provide a satisfying model for the limits and capabilities of natural language, as well as understanding the universality of natural language.

Bibliography

Frank, R. (2003). “Restricting grammatical complexity.” Cognitive Science 28: 669-697.

Gazdar, G. (1988). “Applicability of indexed grammars to natural languages.” U. Reyle and C.

Pullum, G. (1986). “Footloose and context-free.” Natural Language & Linguistic Theory 4(3): 409- 414.

Shieber, S. (1985). “Evidence against the context-freeness of natural language.” Linguistics and Philosophy 8: 333-345.