The goal of this project is to convert a Context Free Grammar into a Strictly 2-Local Tree Grammar and create a corresponding Finite-State Tree Acceptor. In order to do this, Natural Language Tool Kit and a Context Free Grammar were used. NLTK provides a demo CFG, which was used.

A Context Free Grammar is a set of production rules that represents all the possible strings in a language. It consists of a non terminal start symbol *S* and takes the form

*S* -> A  
A -> b

where A is non-terminal and b is terminal. The right side can always replace the left side, and that is why it is referred to as “context free.” CFG’s are used to describe relationships between words in phrases. A language can be described by a CFG if all elements in the Language are allowable by the CFG.

A Strictly 2-Local Tree Grammar can be defined as a set of substrings of the length 2 which occur inside the set of strings of the language (Heinz and Jardine, 2016). In this case, the strings of the language are all possible sentences that can be generated by the CFG. For this project, a bigram grammar was suitable, and fit the criteria of a 2- local grammar.

String : A man walked in the park  
Substrings : {(#,A) (A, man) (man, walked) (walked, in) (in, the) (the, park) (park, #)}

Where # signal possible initial and final words in a sentence of the grammar. A string is a sentence, and the substrings of the sentence are of the size 2, which fit nicely into bigrams. This in turn is a bigram grammar.

The Finite State Tree Acceptor (FSA) is a machine of which the output is binary: acceptable or unacceptable. An FSA consists of a dictionary, possible transitions and states, an initial state, and a final state. This project’s input is a sentence. The FSA will answer as acceptable if the sentence was acceptable by the CFG. The FSA will return unacceptable if a word in the sentence is not part of the lexicon of the CFG, if the sentence does not begin or end on an initial or final word, and or if any the bigrams of the input sentence are not part of the bigram grammar.

The initial intention for this project was to start with creating a CFG from the parsed sentences in the Penn Treebank sample corpus. This Penn Treebank CFG (PTCFG) in Chomsky Normal Form has over 200,000 productions. In order to create bigrams out of the PTCFG, the next step was to first generate all possible sentences that the PTCFG allows. This proved troublesome as a consequence of the number of productions and infinite recursion of the grammar. The problem of infinite recursion could be solved by limiting the depth of the generated sentences. The same problem still occurred, and the program was not able to generate sentences because of the large number of productions. In short, the program would “time out.”

In place of the PTCFG, a small demo CFG called *demo\_grammar* from NLTK was used. The *generate* function from NLTK was then used to generate a list of all the possible sentences from the CFG. The demo CFG has the ability to generate 114 sentences that are grammatical according to this grammar. NLTK already has a ready-made function to obtain bigrams from text. This was not used for this project. A function was created from scratch to extract all bigrams from the generated sentences. This was done by splitting the individual words from each of the sentences, adding initial and final signals (#), and iterating through each sentence to create tuples of the size 2 by adding the 2nd word to the first, the 3rd word to the 2nd and so on. Then duplicate tuples were removed. The resulting list of tuples is the bigram grammar.

A function was created that generates FSA transitions from bigram transitions, in the form of a python dictionary data type, that consists of strings (keys for the python dictionary) and a set (values that are corresponding to the key) in the form of

{a : {b,c,d}, d : {f,g}}

which expresses the transitions such as

a -> b  
a -> c  
a -> d  
d -> f  
d -> g

The input for the FSA is a sentence. The function *chk\_sent* takes an input sentence and outputs whether or not the sentence is acceptable according to the bigram grammar. The function first adds initial and final symbols to the input sentence, then extracts the bigrams from the input sentence. The function then compares the bigrams of the input sentence one by one to the bigram grammar. If any of the words of the sentence are not part of the dictionary, the function returns “unacceptable.” If the initial or final word of the input sentence does not match any of the initial or final words in the bigram grammar, the function will return “unacceptable.” If any of the bigrams of the sentence are not in the bigram grammar, the function returns “unacceptable.”

All sentences which are acceptable by the original demo CFG are acceptable by the FSA. There are problems that arise, because the demo CFG does turn out ungrammatical sentences, which means the FSA will accept ungrammatical sentences, as long as they are allowable by the bigram grammar created from the CFG’s generated sentences. The demo grammar used has only 13 productions with start state S. The CFG also only has 10 words in its lexicon. It turns out nonsensical sentences such as, “A park walked in a man.”

S -> NP VP

NP -> Det N

PP -> P NP

VP -> 'slept'

VP -> 'saw' NP

VP -> 'walked' PP

Det -> 'the'

Det -> 'a'

N -> 'man'

N -> 'park'

N -> 'dog'

P -> 'in'

P -> 'with'

This is the reason that during the first attempt, the PTCFG was used, because it should not generate ungrammatical sentences. Another attempt with a home-made CFG was made, but the problem of infinite recursion caused sentences to have the same adjective repeatedly inside one Adjective Phrase. This can be partially resolved by only allowing a certain depth of sentence, and removing repeated words from the bigrams, but there is still the problem of the program timing out.

In order to only accept grammatical sentences, the original CFG must not generate any ungrammatical sentences. This could be solved if the CFG is induced from grammatical sentences, such as the Penn Treebank.

To test phrases and sentences in the program, the input sentence can be changed in the main code, line 208: input\_sentence. Something that is not included in this program is the use of the NLTK bigram function. Upcoming testing would include the use of the NLTK bigram function to get bigrams from a text, in order to test the FSA with a grammatical bigram grammar.

References:

Adam Jardine and Jeffrey Heinz. **Learning Tier-based Strictly 2-Local Languages**. Transactions of the Association for Computational Linguistics, 4:87--98, April 2016.