Context Free Grammar Development Program
COMP 6751 Project 2 Report

Haochen Zou (40158179)

0. Expectations of originality

I certify that this submission is my original work and meets the Faculty's Expectations of Originality.
Name: Haochen Zou; I.D: 40158179; Date: 2021.10.12

1. Introduction

For the project required in the assignment, implemented a context free grammar that covers different text content and generate a parse tree to represent the syntactic structure.

2. Materials and Methods

2.1. Techniques

2.1.1. FeatureEarleyChartParser

The Earley algorithm is like top-down statement parsing in that it can handle left recursion and does not require CNF transformation. The Earley algorithm fills the line graph from left to right. FeatureEarley-ChartParser is a chart parser in NLTK implementing the Earley parsing algorithm, allowing nonterminal that have features. It uses a lexicon to decide whether a leaf has a given part of speech. This lexicon is encoded as a dictionary that maps each word to a list of parts of speech that word can have.

2.1.2. nltk.draw.tree

A natural language toolkit module for representing hierarchical language structures, such as syntax trees and morphological trees.

2.2. Architecture

In the context free grammar development program, seven operations can be performed:

1. Users enter the text to parse.
2. Give options to save and print the parse tree.
3. Reveal the results of sentences splitting.
4. Show the part of speech tagging.
5. Process name entities and give results.
6. Implement Earley parse to the text and generate parse trees.
7. Display the Earley parsing process progress.

Flow chart of the context free grammar development program is shown in **Figure** 1 below.

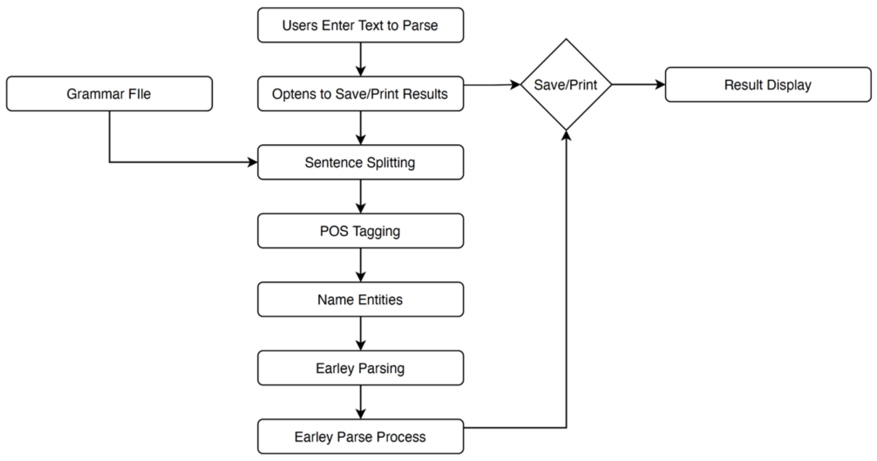
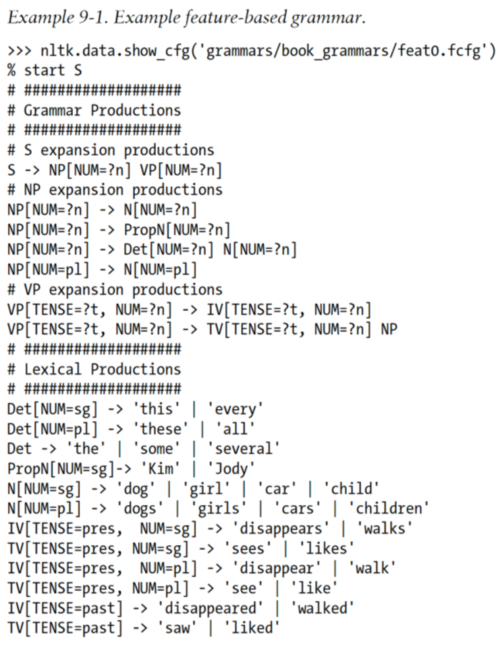


Figure 1. Flow chart of the context free grammar development program.

3. Implementation

3.1. Context Free Grammar

For the project requirement, we need to develop a context free grammar covers the text validation content. In general, when we develop a grammar, it is convenient to put the rules in a file where they can be edited, tested, and revised. We have saved grammar as a file named 'grammar.fcfg' and placed it in the NLTK data directory. We can inspect it as follows in **Figure** 2 from book Natural Language Processing with Python in page 334. Then feature-based grammars are parsed in NLTK using an Earley chart parser.



Example 9-1. Example feature-based grammar.

```
>>> nltk.data.show_cfg('grammars/book_grammars/feat0.fcfg')
% start S
# ###################
# Grammar Productions
# ###################
# S expansion productions
S -> NP[NUM=?n] VP[NUM=?n]
# NP expansion productions
NP[NUM=?n] -> N[NUM=?n]
NP[NUM=?n] -> PropN[NUM=?n]
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
NP[NUM=pl] -> N[NUM=pl]
# VP expansion productions
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t, NUM=?n]
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP
# ###################
# Lexical Productions
# ###################
Det[NUM=sg] -> 'this' | 'every'
Det[NUM=pl] -> 'these' | 'all'
Det -> 'the' | 'some' | 'several'
PropN[NUM=sg]-> 'Kim' | 'Jody'
N[NUM=sg] -> 'dog' | 'girl' | 'car' | 'child'
N[NUM=pl] -> 'dogs' | 'girls' | 'cars' | 'children'
IV[TENSE=pres,  NUM=sg] -> 'disappears' | 'walks'
TV[TENSE=pres, NUM=sg] -> 'sees' | 'likes'
IV[TENSE=pres,  NUM=pl] -> 'disappear' | 'walk'
TV[TENSE=pres, NUM=pl] -> 'see' | 'like'
IV[TENSE=past] -> 'disappeared' | 'walked'
TV[TENSE=past] -> 'saw' | 'liked'
```

Figure 2. Information about example feature-based grammar from the book.

The context free grammar in the project is defined as follows.

```
% start S
# ###################
# Grammar Productions
# ###################
# S expansion productions
S -> NP[NUM=?n] VP[NUM=?n] | CC NP[NUM=?n] VP[NUM=?n] | NP[NUM=?n] VP[NUM=?n] DATE
S -> NP[NUM=?n] VP[NUM=?n] CC TV[NUM=?n] INTRO S
S -> NP[NUM=?n] TV[NUM=?n] INTRO S | NP[NUM=?n] TV[NUM=?n] INTRO S CC IN S
S -> PP COMMA NP[NUM=?n] VP[NUM=?n] | DATE COMMA NP[NUM=?n] VP[NUM=?n] | DATE COMMA PP
COMMA NP[NUM=?n] VP[NUM=?n]
S -> PP COMMA NP[NUM=?n] TV[NUM=?n] INTRO S | DATE COMMA NP[NUM=?n] TV[NUM=?n] INTRO S
S -> PP COMMA DATE COMMA NP[NUM=?n] TV[NUM=?n] NP[NUM=?n] INTRO S
S -> RB COMMA NP[NUM=?n] VP[NUM=?n]
# NP expansion productions
NP[NUM=?n] -> DT[NUM=?n] Nom[NUM=?n] | DT[NUM=?n] Nom[NUM=?n] PP | Nom[NUM=?n] PP
NP[NUM=?n] -> PRP | Nom[NUM=?n]
NP[NUM=?n] -> PRP NP[NUM=?n] | PRP DATE PP
NP[NUM=?n] -> DT[NUM=?n] JJ NP[NUM=?n]
NP[NUM=?n] -> N[NUM=?n] CC NP[NUM=?n] | NNP[NUM=?n] CC NP[NUM=?n]
Nom[NUM=?n] -> N Nom[NUM=?n] | N | NNP[NUM=?n]
# VP expansion productions
VP[TENSE=?t, NUM=?n] -> VP[TENSE=?t, NUM=?n] PP
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP | RB TV[TENSE=?t, NUM=?n] NP | MD[TENSE=?t]
TV[TENSE=inf] NP RB
VP[TENSE=?t, NUM=?n] -> MD[TENSE=?t] TV[TENSE=inf] NP | MD[TENSE=?t] AUX[TENSE=inf] JJ |
AUX[TENSE=?t] JJ | MD[TENSE=?t] AUX[TENSE=inf] | MD[TENSE=?t] IV[TENSE=inf]
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t] TO VP[TENSE=inf]
VP[TENSE=?t, NUM=?n] -> VP[TENSE=?t, NUM=?n] PP | IV[TENSE=?t]
# PP expansion productions
PP -> IN NP | IN DATE
```

```
# ###################
# Lexical Productions
# ###################

DT[NUM=?n] -> Det[NUM=?n]
Det[NUM=sg] -> 'an' | 'a'
Det[NUM=pl] -> 'these' | 'some' | 'all'
Det[NUM=?n] -> 'the'

AUX[TENSE=inf] -> 'be'
AUX[TENSE=past] -> 'was'

MD[TENSE=inf] -> 'will'
MD[TENSE=past] -> 'would'

COMMA -> ','

NNP[NUM=?n] -> PropN[NUM=?n]
PropN[NUM=sg] -> 'John' | 'Mary' | "John O'Malley" | "O'Malley" | 'Sue' | "OMalley"

N[NUM=sg] -> NN
N[NUM=pl] -> NNS
NN -> 'apple' | 'table' | 'fridge' | 'office' | 'refrigerator' | 'desk' | 'colleague' | 'replacement' | 're-placement' |
'treat' | 'day'

NNS -> 'apples'

PRP -> 'his' | 'he' | 'He' | 'it' | 'It' | 'her' | 'them' | 'she' | 'I' | 'you'

IN -> P

P -> 'at' | 'in' | 'from' | 'to' | 'on' | 'that' | 'with' | 'for'
P -> 'On'

TO -> 'to'

CC -> 'and' | 'but' | 'But'

JJ -> 'Last' | 'crunchy' | 'sick' | 'last'

RB -> 'finally' | 'both' | 'Finally'

TV[TENSE=pres, NUM=sg] -> 'puts'
TV[TENSE=pres, NUM=pl] -> 'put'
TV[TENSE=inf] -> 'put' | 'eat' | 'share' | 'delight' | 'promise'
TV[TENSE=past] -> 'ate' | 'took' | 'promised' | 'anticipated' | 'said' | 'put'

IV[TENSE=past] -> 'intended' | 'ate'

DATE -> YEAR SEP MONTH_NUM SEP DAY | MONTH_STR DAY SEP YEAR | MONTH_STR DAY | MONTH_STR
YEAR | PP COMMA NP[NUM=?n] TV[NUM=?n] INTRO S  MONTH_STR NN_NUM YEAR | MONTH_STR NN_NUM
SEP YEAR | MONTH_STR NN_NUM | YEAR | MONTH_STR YEAR | WEEK | JJ WEEK

SEP -> "/" | "-" | ","

NN_NUM -> "1st" | "2nd" | "3rd" | "4th" | "5th" | "6th" | "7th" | "8th" | "9th" | "10th" | "11th" | "12th" | "13th" |
"14th" | "15th" | "16th" | "17th" | "18th" | "19th" | "20th" | "21st" | "22nd" | "23rd" | "24th" | "25th" | "26th" |
"27th" | "28th" | "29th" | "30th" | "31st"

YEAR -> '1900' | '1901' | '1902' | '1903' | '1904' | '1905' | '1906' | '1907' | '1908' | '1909' | '1910' | '1911' |
'1912' | '1913' | '1914' | '1915' | '1916' | '1917' | '1918' | '1919' | '1920' | '1921' | '1922' | '1923' | '1924' | '1925'
| '1926' | '1927' | '1928' | '1929' | '1930' | '1931' | '1932' | '1933' | '1934' | '1935' | '1936' | '1937' | '1938' |
'1939' | '1940' | '1941' | '1942' | '1943' | '1944' | '1945' | '1946' | '1947' | '1948' | '1949' | '1950' | '1951' | '1952'
| '1953' | '1954' | '1955' | '1956' | '1957' | '1958' | '1959' | '1960' | '1961' | '1962' | '1963' | '1964' | '1965' |
'1966' | '1967' | '1968' | '1969' | '1970' | '1971' | '1972' | '1973' | '1974' | '1975' | '1976' | '1977' | '1978' | '1979'
```

```
| '1980' | '1981' | '1982' | '1983' | '1984' | '1985' | '1986' | '1987' | '1988' | '1989' | '1990' | '1991' | '1992' |
'1993' | '1994' | '1995' | '1996' | '1997' | '1998' | '1999' | '2000' | '2001' | '2002' | '2003' | '2004' | '2005' | '2006'
| '2007' | '2008' | '2009' | '2010' | '2011' | '2012' | '2013' | '2014' | '2015' | '2016' | '2017' | '2018' | '2019' |
'2020' | '2021' | '2022' | '2023' | '2024' | '2025' | '2026' | '2027' | '2028' | '2029' | '2030'

WEEK -> 'week' | 'Monday' | 'Tuesday' | 'Wednesday' | 'Thursday' | 'Friday' | 'Saturday' | 'Sunday'

MONTH_STR -> "January" | "February" | "March" | "April" | "May" | "June" | "July" | "August" | "September" |
"October" | "November" | "December"
MONTH_NUM -> '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12'

DAY -> '00' | '01' | '02' | '03' | '04' | '05' | '06' | '07' | '08' | '09' | '10' | '11' | '12' | '13' | '14' | '15' | '16' | '17' | '18' |
'19' | '20' | '21' | '22' | '23' | '24' | '25' | '26' | '27' | '28' | '29' | '30' | '31' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

INTRO -> 'that'
```

3.2. Sentence Splitting

Sentence splitting as for tokenization, start with a NLTK module. Standard sentence tokenizer **sent_tokenize** return a sentence-tokenized copy of text file using NLTK's recommended sentence tokenizer.

In the project, the sentence splitting part is written in the code shown below.

```
body_sentences = nltk.sent_tokenize(body)
print('\n【Sentences Splitting】')
print(body_sentences)
```

3.3. POS Tagging

In this project, A part-of-speech tagger, or POS tagger, processes a sequence of words, and attaches a part of speech tag to each word. Next step is representing tagged tokens and reading tagged corpora. Finally mapping words to properties using python dictionaries. The code about this module is displayed as follows.

```
pos_tag: List[List[str]] = list()
token_list: List[List[str]] = list()
name_entity: Set[str] = set()
for text_sentence in sentence:
    # word tokenization
    words = word_tokenize(text_sentence)
    # name entity
    words, name_entity = name_entity_module(words)
    token_list.append(words[:-1])   # omit the last period
    name_entity = name_entity.union(name_entity)
    # POS tagging
    pos_tag.append(pos_tagging(words))
print('\n【POS Tagging】')
print(pos_tag)
```

3.4. Name Entity

Name entity processing is implemented in the part-of-speech tagging part, relevant code shown below.

```
print('\n【Name Entities】')
print(name_entity)
```

3.5. Earley Parse

Earley parser in NLTK of this project is designed based on the context free grammar we develop. The text user input will be parsed then generate a parsing tree. Code about this part displayed as follows.

```
def parse_print_text(self, token_lists: List[List[str]]) -> None:
    self.parser.save_result()
    for ts in token_lists:
        self.parser.parse(ts)

print('\n【Earley Parsing】')
self.parse_print_text(token_list)
```

3.6. Earley Parse Process

In this project, we develop an Earley parse process progress to display how the text being parsed by the program in the context free grammar we designed to proofread and check the results above.

The code about Earley parse process progress is shown below.

```
with open("text.txt", "r") as f:
  data = f.read()
text_sentence = data
sentence_without_punctuation = re.sub(r'[^\w\s]','',text_sentence)
tokens = sentence_without_punctuation.split()
cp = parse.load_parser('grammar.fcfg', trace=1)

print('\n【Earley Parse Process】')
trees = cp.parse(tokens)
print(trees)
```

4. Errors and Limitations

Please check the Demo File.

5. Reference

Natural Language Processingwith Python, Steven Bird, Ewan Klein, and Edward Loper
https://docs.huihoo.com/nltk/0.9.5/api/nltk.draw.tree.TreeView-class.html
https://www.nltk.org/_modules/nltk/draw/tree.html
http://www.nltk.org/howto/featgram.html