Name: Bhavesh Kewalramani
Roll No.: A-25

Practical No. 7

Theory

Background: Parser uses a CFG(Context-free-Grammar) to validate the input string and produce output for the next phase of the compiler. Output could be either a parse tree or an abstract syntax tree. Now to interleave semantic analysis with the syntax analysis phase of the compiler, we use Syntax Directed Translation.

Definition

Syntax Directed Translation has augmented rules to the grammar that facilitate semantic analysis. SDT involves passing information bottom-up and/or top-down the parse tree in form of attributes attached to the nodes. Syntax-directed translation rules use 1) lexical values of nodes, 2) constants & 3) attributes associated with the non-terminals in their definitions.

The general approach to Syntax-Directed Translation is to construct a parse tree or syntax tree and compute the values of attributes at the nodes of the tree by visiting them in some order. In many cases, translation can be done during parsing without building an explicit tree.

Example

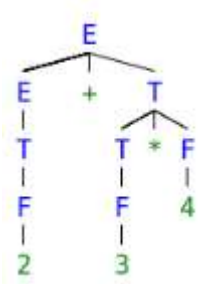
```
E -> E+T | T

T -> T*F | F

F -> INTLIT
```

This is a grammar to syntactically validate an expression having additions and multiplications in it. Now, to carry out semantic analysis we will augment SDT rules to this grammar, in order to pass some information up the parse tree and check for semantic errors, if any. In this example, we will focus on the evaluation of the given expression, as we don't have any semantic assertions to check in this very basic example.

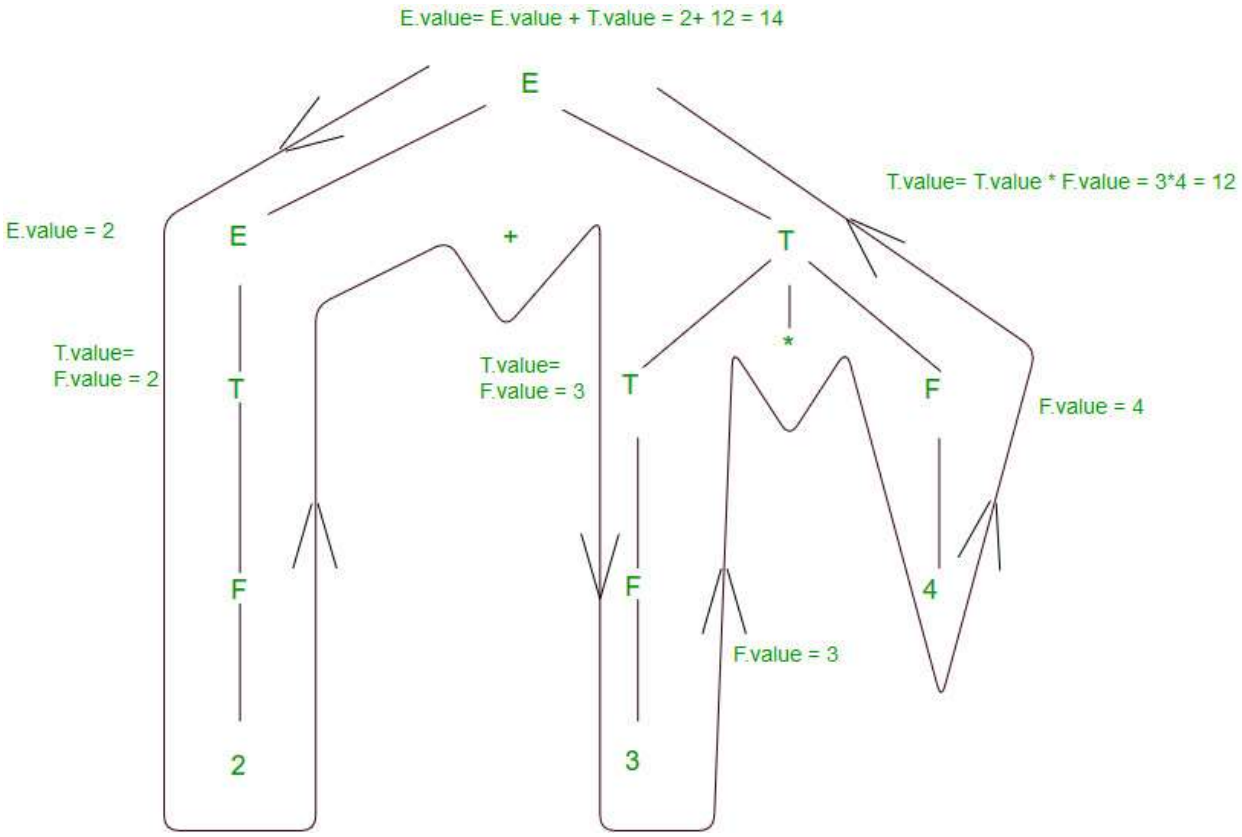
```
E -> E+T     { E.val = E.val + T.val }   PR#1

E -> T       { E.val = T.val }           PR#2

T -> T*F     { T.val = T.val * F.val }   PR#3

T -> F       { T.val = F.val }           PR#4

F -> INTLIT  { F.val = INTLIT.lexval }   PR#5
```

For understanding translation rules further, we take the first SDT augmented to [E -> E+T] production rule. The translation rule in consideration has val as an attribute for both the non-terminals – E & T. Right-hand side of the translation rule corresponds to attribute values of right-side nodes of the production rule and vice-versa. Generalizing, SDT are augmented rules to a CFG that associate 1) set of attributes to every node of the grammar and 2) set of translation rules to every production rule using attributes, constants, and lexical values.

Let's take a string to see how semantic analysis happens – S = 2+3*4. Parse tree corresponding to S would be



To evaluate translation rules, we can employ one depth-first search traversal on the parse tree. This is possible only because SDT rules don't impose any specific order on evaluation until children's attributes are computed before parents for a grammar having all synthesized attributes. Otherwise, we would have to figure out the best-suited plan to traverse through the parse tree and evaluate all the attributes in one or more traversals. For better understanding, we will move bottom-up in the left to right fashion for computing the translation rules of our example.



The above diagram shows how semantic analysis could happen. The flow of information happens bottom-up and all the children's attributes are computed before parents, as discussed above. Right-hand side nodes are sometimes annotated with subscript 1 to distinguish between children and parents.
Additional Information
Synthesized Attributes are such attributes that depend only on the attribute values of children nodes.
Thus [E -> E+T { E.val = E.val + T.val }] has a synthesized attribute val corresponding to node E. If all the semantic attributes in an augmented grammar are synthesized, one depth-first search traversal in any order is sufficient for the semantic analysis phase.
Inherited Attributes are such attributes that depend on parent and/or sibling's attributes.
Thus [Ep -> E+T { Ep.val = E.val + T.val, T.val = Ep.val }], where E & Ep are same production symbols annotated to differentiate between parent and child, has an inherited attribute val corresponding to node T.

Practicals

Aim:

(A) Read the Research paper provided to understand the use of SDTS in language translation from English to Hindi.

(B) Write a program to implement tokenizer, syntax analyzer, and to perform syntax directed translation for conversion from English to Hindi as per the research paper. Implement all the functions and extend the procedure to generate correct Hindi sentence.

(A)

Hindi machine translation is a text translation From one language to another through a non-human computer Participation. MT is a subset of natural language processing Intended to change natural language texts and speeches I'm using different software. This change helps us To overcome the language barrier. Even in the invention of Can overcome the technical and cultural barriers of MT Allows easy and frequent knowledge sharing. the My current work is trying to set up automatic translation A system for converting text from English to Hindi. Several Work on machine translation was also done. Introduced here. First, get the text as an English input. Save it to a file, extract words and punctuation from it, Store them in an array, then group and search for words Their meanings related to the sentence and convert it to Target language, that is, H. Hindi. In this translation, In some topics such as word order, word meaning, ambiguity, etc. And idioms. Finally, if you get the text in Hindi, check it For the accuracy of the grammatical rules of the language.

Machine translation is a process that helps in conversion of the given text form to the required target language. Thus, MT helps in overcoming lingual barriers and technological barriers. Also an MT system can replace a human translator which helps in minimizing our precious time and expenses too. In this paper, English to Hindi Machine Translator System for watchers has been developed. In this we are using example based approach. We have trained our system by taking several sample texts in English and their corresponding Hindi sentences. Database is used to store text in source and target languages which is referred during translation process. Also we maintain an Encyclopedia and Dictionary to store words which have no meaning in Hindi are grouped under former category and the latter stores the remaining. Thus, the proposed system aims at translating text from English to Hindi using stepwise procedure discussed by algorithm.

(B)
Program:

```
import spacy
import csv
import pandas as pd
import re
from indic_transliteration import sanscript
from indic_transliteration.sanscript import transliterate
from collections import OrderedDict
from textblob import TextBlob

def getGrammar(rule,start):
```

```
    terminal=[]
    nonterminal=[]
    r=[]
    k=rule.keys()
    for i in rule.values():
        for j in i:
            for a in list(str(j).split(' ')):
                if a not in k:
                    if a not in r:
                        r.append(a)

    r.append("$")
    t=dict()
    for i in rule.keys():
        t[i]=dict()
        for j in r:
            t[i].update({j:set()})

    return rule,start,r,t

def Calculate_First(s,rule,v):
    s=list(str(s))
    if s[0] in v:
        return set([s[0]])
    elif s[0] == '`':
        return set(["`"])
    else:
        res = set()
        for j in rule[s[0]]:
            h = Calculate_First(list(str(j).split(' ')),rule,v)
            res.update(set(h))
        if len(s) == 1:
            return res
        else:
            if '`' in res:
                res.remove('`')
                return res.union(Calculate_First(list(s.split(' '))[1:],rule,v))
            return res


def getValue(v,rule):
    for key, value in rule.items():
        if value == v:
            return key
```

```
def Calculate_Follow(s,rule,v,start):
    res = set()
    if s == start:
        res = set(['$'])
    for i in rule.values():
        for j in range(len(i)):
            l = list(i[j].split(' '))
            for k in range(len(l)):
                if l[k] == s:
                    if k == len(l) - 1:
                        if getValue(i,rule) == s:
                            continue
                        else:
                            res.update(set(Calculate_Follow(getValue(i,rule),rule,v,start)))
                    else:
                        c = set(Calculate_First(l[k + 1:],rule,v))
                        if '`' in c:
                            c.remove('`')
                            fol = Calculate_Follow(getValue(i,rule),rule,v,start)
                            c.update(fol)
                            res.update(c)
                        else:
                            res.update(set(c))
    return res


def parseString(string,t,start):
    string.append('$')
    stk = ['$', start]
    while not len(stk) == 0:
        top = stk[-1]
        stk.pop()
        if string[0] == '$' and top == '$' and len(stk) == 0:
            return 0
        elif (string[0] == '$' and len(stk) != 0) or (string[0] != '$' and len(stk) == 0):
            return 1
        elif top == string[0]:
            string = string[1:]
        else:
            for i in t[top][string[0]]:
                l = list(i.split(' '))
            for j in string[::-1]:
                stk.append(j)


nlp = spacy.load("en_core_web_sm")
```

```
s=input("Enter a string you want to translate : ")
s=s.lower()

res = re.sub(r'[^\w\s]', '', s)
res=res.strip()

str1 = list(res.split(" "))
str2=res

doc = nlp(res)
a=[]
b=[]
for token in doc:
    b.append([token,token.pos_])
    a.append((token,token.pos_))


words, tags = zip(*a)


d1={}
for i in range(len(tags)):
    e=[]
    for j in range(len(b)):
        if tags[i]==b[j][1]:
            e.append(b[j][0])
    d1.update({tags[i]:list(set(e))})



ke = []
for key in d1.keys():
    ke.append(key)

d11 = {"N":[], "P":[] ,"V": [],"ADVERB":
[],"PROPN":[],"ADJ":[],"ADP":[],"D":[],"CONJ":[],"INTJ":[],"PN":[]}

rules = {"S": ["NP VP"], "NP": ["P", "PN", "D N"], "VP": ["V NP"],
"N":[],"P":[],"D":[],"V":[],"PN":[]}

if "AUX" in d1.keys():
    rules["V"]+=d1["AUX"]
    d1.pop("AUX")
if "CCONJ" in d1.keys():
    rules["CONJ"]+=d1["CCONJ"]
    d1.pop("CCONJ")
```

```
if "SCONJ" in d1.keys():
    rules["CONJ"]+=d1["SCONJ"]
    d1.pop("SCONJ")

for k,v in d1.items():
    if k=="NOUN":
        rules["N"]=d1[k]
    if k=="PRON":
        rules["P"]=d1[k]
    if k=="PROPN":
        rules["PN"]=d1[k]
    if k=="DET":
        rules["D"]=d1[k]

d1=d11


res=s.split()

rules.update(d1)


start="S"

d,start,r,t=getGrammar(rules,start)

print()

for i in d.keys():
    for rule in d[i]:
        fir = Calculate_First(rule,d,r)
        if '`' in fir:
            fol = Calculate_Follow(i,d,r,start)
            for j in fol:
                if j != '$':
                    t[i][j].add(rule)
            if '$' in fol:
                t[i]['$'].add(rule)
        else:
            for j in fir:
                t[i][j].add(rule)

flagg = parseString(res,t,start)

if flagg==1:
    keys = []
```

```
    ans=""

    dict11={"is":"है","a":"एक","they":"वे","them":"उन्हें","running":"दौड़ रहा","sleeping":"सो रहा",
            "eating":"खा रहा","win":"जीत","lose":"हार"}

    for key in dict11.keys():
        keys.append(key)

    for i in range(len(str1)):
        if str1[i] in keys:
            ans=ans+dict11[str1[i]]+" "
        else:
#             ans=ans+transliterate(i, sanscript.ITRANS, sanscript.DEVANAGARI)+" "
            ttt=str1[i]
            blob = TextBlob(ttt)
            ans+=str(blob.translate(from_lang='en',to='hi'))+" "

    print("Translated String : ",end=" ")
    print(ans)
else:
    print("NOT POSSIBLE")
```

Output:

```
Enter a string you want to translate : John is running

Translated String :  जॉन है दौड़ रहा
```

```
# John is a boy

Enter a string you want to translate : John is a boy

Translated String :  जॉन है एक लड़का
```

```
Enter a string you want to translate : eating is John

NOT POSSIBLE
```