Ilieș Iulia
Ionașcu Iulia

Parser - documentation

https://github.com/iuliaaai/LFTC
The parser works on g1.txt as well as on g2.txt (with PIF).

Lab6: Recursive descendant

The main idea of this strategy is based on a backtracking approach: starting from the starting symbol, we take each production one at a time and try to build the desired sequence. If at one moment we realize that we deviated from the sequence, we go back and try again. In order to achieve this, we keep track of our current state (normal, error, back and final), our position in the sequence, a work stack keeping track of how we build our tree and an input stack, keeping track of the tree-area that has not yet been tried out.
The algorithm is formed of 5 scenarios:

- Expand: head of input stack is a non-terminal, we can replace it using a production and continue
- Advance: head of input stack is a terminal = current symbol from input
- Momentary insuccess: head of input stack is a terminal ≠ current symbol from input
- Back: head of working stack is a terminal
- Another try: head of working stack is a non-terminal
- Success: input stack is empty, index is equal to sequence length

```
def recursive_descendant(grammar, sequence, config):
   while config.state != State.FINAL and config.state != State.ERROR:
       if config.state == State.NORMAL:
           if len(config.input_stack) == 0 and config.index == len(sequence):
               success(config)
           elif len(config.input_stack) == 0:
               momentary_insuccess(config)
           else:
               if config.input_stack[0] in grammar.N:
                   expand(config, grammar)
               else:
                   if config.index == len(sequence):
                       momentary_insuccess(config)
                   elif config.input_stack[0] == 'E':
                       config.work_stack.append('E')
                       config.input_stack = config.input_stack[1:]
                   elif config.input_stack[0] == sequence[config.index]:
                       advance(config)
                   else:
                       momentary_insuccess(config)
       else:
           if config.state == State.BACK:
               if config.work_stack[-1] in grammar.E:
```

```
                    if config.work_stack[-1] == 'E':
                        terminal = config.work_stack.pop(-1)
                        config.input_stack = [terminal] + config.input_stack
                    else:
                        back(config)
                else:
                    another_try(config, grammar)

    prod_rules = []
    print(config.work_stack)
    if config.state == State.ERROR:
        return False, []
    else:
        for prod in config.work_stack:
            if len(prod) > 1:
                if prod[0] in grammar.P.keys():
                    if prod[1] in grammar.P[prod[0]]:
                        prod_rules.append(prod)

    return True, prod_rules
```

The rest of the code is attached to lab6.

Lab7: Parser

After applying the recursive descendant algorithm on a sequence, we can access its work stack. In the end, this is formed of the productions applied in order to achieve the desired sequence. Our goal was to transform the stack into a tree structure which is done by the parse_tree function. We go through the list of productions and build the tree by creating nodes from each terminal and nonterminal with its productions. We put the indexes in depth. Then we parse again the work_stack and we insert in a stack the index of the father as many times as the number of its children. Whenever there is a new node we insert on position 0. This will help us retrieve the father for each node even if it's more of them. For each node we keep its right sibling or -1 in case there isn't one. If the recursive descendant fails, we return an empty array and we don't enter the parse_tree function.

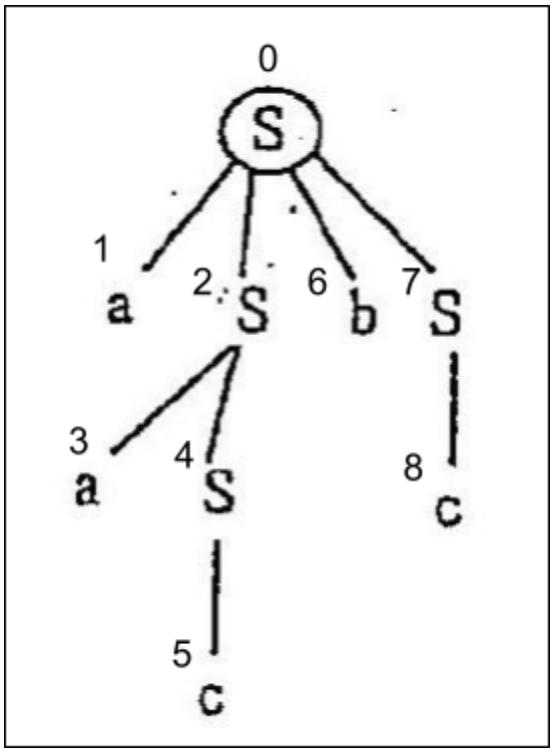
A tree node has the following properties:

- Index: value chosen by parsing in depth from left to right
- Father: index of its parent symbol
- Right Sibling
- Value
- Production: -1 for terminals, or the production used for a non-terminal

Exemple(g3.out)
index | value | father | sibling

Ilieș Iulia
Ionașcu Iulia

0 S -1 -1
1 a 0 2
2 S 0 6
3 a 2 4
4 S 2 -1
5 c 4 -1
6 b 0 7
7 S 0 -1
8 c 7 -1




```
class Node:
    def _init_(self, value, index):
        self.index = index
        self.father = -1
        self.sibling = -1
        self.value = value
        self.production = -1

    def _str_(self):
        return str(self.value) + " " + str(self.father) + " " +
str(self.sibling)


class Parser:

    def _init_(self, grammar):
        self.grammar = grammar
```

```
        self.config = Configuration(grammar.S)
        self.iteration = 0
        self.tree = []
        self.words = []

    def parse_tree(self):
        fathers = [-1]

        for pos in range(0, len(self.config.work_stack)):
            if type(self.config.work_stack[pos]) == tuple:  # non terminal with
production
                self.tree.append(Node(self.config.work_stack[pos][0], pos))
                self.tree[pos].production = self.config.work_stack[pos][1]
            else:
                self.tree.append(Node(self.config.work_stack[pos], pos))  #
terminal

        for pos in range(0, len(self.config.work_stack)):
            if type(self.config.work_stack[pos]) == tuple:
                self.tree[pos].father = fathers[0]
                fathers = fathers[1:]
                len_production = len(self.config.work_stack[pos][1])
                child_indexes = []
                for i in range(0, len_production):
                    child_indexes.append(pos + i + 1)
                    fathers.insert(0, pos)
                for i in range(0, len_production):
                    if self.tree[child_indexes[i]].production != -1:
                        offset =
self.get_production_depth_offset(child_indexes[i])
                        for j in range(i + 1, len_production):
                            child_indexes[j] += offset
                for i in range(0, len_production - 1):
                    self.tree[child_indexes[i]].sibling = child_indexes[i + 1]
            else:
                self.tree[pos].father = fathers[0]
                fathers = fathers[1:]

    def get_production_depth_offset(self, index):
        prod_rhs= self.config.work_stack[index][1]
        len_production = len(prod_rhs)
        offset = len_production
        for i in range(1, len_production + 1):
            if type(self.config.work_stack[index + i]) == tuple:
                offset += self.get_production_depth_offset(index + i)
        return offset

    def write_tree_to_file(self, filename):
        file = open(filename + ".out", "w")
```

```
        file.write("index | value | father | sibling\n")

        for pos in range(0, len(self.config.work_stack)):
            node = self.tree[pos]
            file.write(str(pos) + " " + str(node) + "\n")
        file.close()
```

```
def isIdentifier(token):
    return re.match(r'^[a-zA-Z]([a-zA-Z]|[0-9])*$', token) is not None


def isConstant(token):
    return
re.match(r'^(0|[+-]?[1-9][0-9]*)$|^(0|[+-]?[1-9][0-9]*),[0-9]*|^`.`$|^`.*`$',
token) is not None


def isStringConstant(token):
    return re.match(r"^'.'$|^'.*'$", token) is not None


def buildSequenceFromPIF(filename):
    sequence = []
    with open(filename, 'r') as file:
        for line in file:
            elem = line.split(":")[0].strip()
            type = line.split(",")[-1].strip()
            if type == '0':
                if isStringConstant(elem):
                    sequence.append('string')
                elif isConstant(elem):
                    sequence.append('integer')
                else:
                    sequence.append('constant')
            elif type == '1':
                sequence.append('identifier')
            else:
                sequence.append(elem)
    print(sequence)
    return sequence
```

g1.txt:

```
N = S A B C
E = 0 1
S = S
```

Ilieș Iulia
Ionașcu Iulia

```
P =
  S -> 0 B | 1 A
  A -> 0 | 0 S | 1 A A
  B -> 1 | 1 S | 0 B B
```

g1.out:

```
index | value | parent | sibling
0 S -1 -1
1 1 0 2
2 A 0 -1
3 1 2 4
4 A 2 6
5 0 4 -1
6 A 2 -1
7 0 6 8
8 S 6 -1
9 0 8 10
10 B 8 -1
11 1 10 12
12 S 10 -1
13 1 12 14
14 A 12 -1
15 1 14 16
16 A 14 18
17 0 16 -1
18 A 14 -1
19 0 18 -1
```

g2.txt:

```
N = program tempDecl declList declaration variableDeclaration constDeclaration
type1 arrayDecl type stmtList stmt simplStmt assignStmt expression term factor
ioStmt stringExp structStmt ifStmt tempElifStmt elseStmt elifStmt whileStmt
condition relation
E = + - * / < <= = >= > ?: == % ! != @ [ ] { } # “ ” ‘ ’ ; ( ) func Int String
Bool Char if elif else let var ret True False read print loop GO STOP
identifier constant integer string boolean E
S = program
P =
program -> GO tempDecl STOP
tempDecl -> E | declList tempDecl | stmtList tempDecl
declList -> declaration | declaration declList
declaration -> variableDeclaration | constDeclaration
variableDeclaration -> var identifier @ type = expression ; | var identifier @
type ;
constDeclaration -> let identifier @ type = expression ;
type1 -> Bool | Int | Char | String
arrayDecl -> [ type1 ]
type -> type1 | arrayDecl
```

Ilieș Iulia
Ionașcu Iulia

```
stmtList -> stmt | stmt stmtList
stmt -> simplStmt | structStmt
simplStmt -> assignStmt | ioStmt
assignStmt -> identifier = expression ;
expression -> term + expression | term - expression | term | boolean
term -> factor * term | factor / term | factor % term | factor
factor -> ( expression ) | identifier | integer
ioStmt -> read ( identifier ) ; | print ( stringExp ) ;
stringExp -> string | identifier
structStmt -> ifStmt | whileStmt
ifStmt -> if condition { stmtList } tempElifStmt | if condition { stmtList }
tempElifStmt elseStmt
tempElifStmt -> E | tempElifStmt elifStmt
elseStmt -> else { stmtList }
elifStmt -> elif condition { stmtList }
whileStmt -> loop condition { stmtList }
condition -> expression relation expression
relation -> < | <= | = | >= | > | !=
```

g2.out:

```
index | value | parent | sibling
0 program -1 -1
1 GO 0 2
2 tempDecl 0 24
3 declList 2 14
4 declaration 3 -1
5 variableDeclaration 4 -1
6 var 5 7
7 identifier 5 8
8 @ 5 9
9 type 5 12
10 type1 9 -1
11 Int 10 -1
12 ; 5 -1
13 tempDecl 2 -1
14 stmtList 13 23
15 stmt 14 -1
16 simplStmt 15 -1
17 ioStmt 16 -1
18 read 17 19
19 ( 17 20
20 identifier 17 21
21 ) 17 22
22 ; 17 -1
23 tempDecl 13 -1
24 declList 23 37
25 declaration 24 -1
26 constDeclaration 25 -1
```

Ilieș Iulia
Ionașcu Iulia

```
27 let 26 28
28 identifier 26 29
29 @ 26 30
30 type 26 33
31 type1 30 -1
32 Int 31 -1
33 = 26 34
34 expression 26 40
35 term 34 -1
36 factor 35 38
37 identifier 36 -1
38 % 35 39
39 term 35 -1
40 factor 39 -1
41 integer 40 -1
42 ; 26 -1
43 tempDecl 23 -1
44 stmtList 43 54
45 stmt 44 -1
46 simplStmt 45 -1
47 ioStmt 46 -1
48 print 47 49
49 ( 47 50
50 stringExp 47 52
51 identifier 50 -1
52 ) 47 53
53 ; 47 -1
54 tempDecl 43 -1
55 stmtList 54 65
56 stmt 55 -1
57 simplStmt 56 -1
58 ioStmt 57 -1
59 print 58 60
60 ( 58 61
61 stringExp 58 63
62 string 61 -1
63 ) 58 64
64 ; 58 -1
65 tempDecl 54 -1
66 stmtList 65 76
67 stmt 66 -1
68 simplStmt 67 -1
69 ioStmt 68 -1
70 print 69 71
71 ( 69 72
72 stringExp 69 74
73 string 72 -1
74 ) 69 75
75 ; 69 -1
```

```
76 tempDecl 65 -1
77 E 76 -1
78 STOP 0 -1
```

pif2.txt:

```
GO : (-1, -1), 18
var : (-1, -1), 11
n : (15, 0), 1
@ : (-1, -1), 35
Int : (-1, -1), 3
; : (-1, -1), 34
read : (-1, -1), 15
( : (-1, -1), 40
n : (15, 0), 1
) : (-1, -1), 41
; : (-1, -1), 34
let : (-1, -1), 10
lastDigit : (2, 0), 1
@ : (-1, -1), 35
Int : (-1, -1), 3
= : (-1, -1), 26
n : (15, 0), 1
% : (-1, -1), 31
10 : (2, 1), 0
; : (-1, -1), 34
print : (-1, -1), 16
( : (-1, -1), 40
lastDigit : (2, 0), 1
) : (-1, -1), 41
; : (-1, -1), 34
print : (-1, -1), 16
( : (-1, -1), 40
'hello there' : (9, 0), 0
) : (-1, -1), 41
; : (-1, -1), 34
print : (-1, -1), 16
( : (-1, -1), 40
'a' : (4, 0), 0
) : (-1, -1), 41
; : (-1, -1), 34
STOP : (-1, -1), 19
```

g3.txt - from Mrs. Motogna's book, page 47

```
N = S
E = a b c
S = S
P =
```

Ilieș Iulia
Ionașcu Iulia

```
  S -> a S b S | a S | c
```

g3.out:

```
index | value | parent | sibling
0 S -1 -1
1 a 0 2
2 S 0 6
3 a 2 4
4 S 2 -1
5 c 4 -1
6 b 0 7
7 S 0 -1
8 c 7 -1
```