ECOTE

Semester: 21L

Author: Jan Świerczyński 293660

Subject: Top-Down parser with backtracking.

I. General overview and assumptions

Task:

Write a Top-Down parser with backtracking.

Overview:

The main purpose of this project is to create a program which could give general view on how the Top-down parser tree is handling given CFG and strings. Parsing technique is recursive. Top-Down parsing attempts to build a parse tree from root to the last leaves. Parser would start from the start symbol “S” and proceed to string. It follows leftmost derivation. Backtracking is the algorithm that replaces leaves in a tree, that were assigned wrongly but were assigned due to the leftmost algorithm.

Example of backtracking:

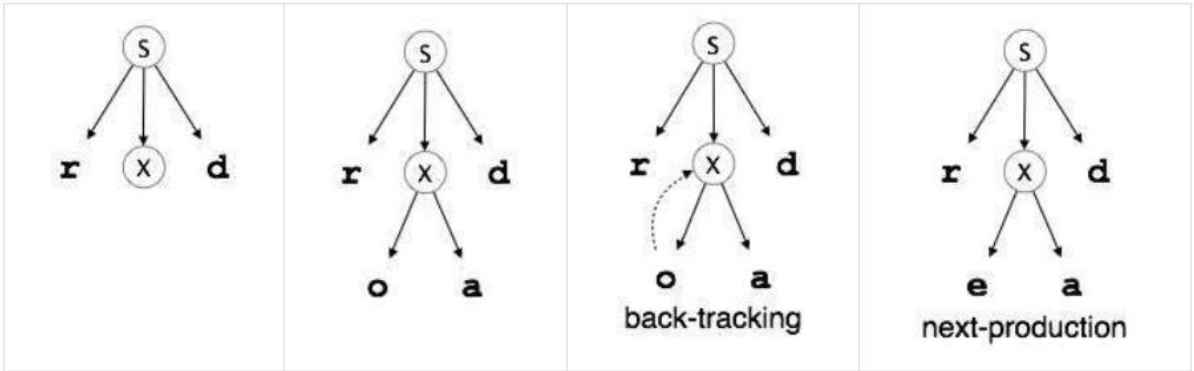
CFG:

S -> rXd | rZd

X -> oa | ea

Z -> ai

Input string: “read”



II. Functional requirements

- Length of the input string should not have limits,
- "$" defines empty string,
- Input string can be a single "$" character,
- It cannot contain any other character together with "$",
- If it does the error message would appear,
- Program working in a terminal,
- The input string would be entered through the console,
- CFG would be given in a *.txt file,
- Capital letters would be treated as non-terminal alphabet,
- Small letters, numbers, rest of special characters, would be treated as terminal alphabet,
- *.txt files with CFG should be written in a specific way (described in **III. Implementation**),
- Path to the CFG file as the 2nd input to the program,
- Program should check whether given string is a part of the CFG,
- TUI representation of string, as a tree, if CFG contains such string,
- If string is not in a CFG program prompts adequate message,
- Root node in non-terminal alphabet would have "S" letter,
- Every CFG without "S" in its alphabet would cause an error.

III. Implementation

General architecture

The program would be written in python3. The main idea behind it is to create a non-binary tree using given string and set of rules of CFG, from *.txt file. Inputted string would always be read from left to right.

First node created would always have a non-terminal value, capital letter "S". Thanks to that assumption the program will know that this node is a root one (Top-down parser builds a parsing tree from root to the leaves). In next step the program would create "S" node's children, using data from *.txt file. There could be a situation that "S" node would have a non-terminal children node, i.e. "a" as leftmost node, or rightmost. If so, the program would check if first (leftmost node) or last (rightmost node) character of the string is equal to the non-terminal value of the leaf. Every node with non-terminal value would be treated as a leaf, there should be no possibility for them to have children nodes. Dependable on the result of the check, next node is created, according to CFG rules, or next production rule of current node is being executed, according to CFG rules as well.

How it works:

1. Read and save string.
2. Check its correctness.
3. Read and save, as an dictionary of strings, the CFG from file.
4. Check its correctness
5. Create a Node, using data from dictionary.
6. Node becomes leftmost one.
7. Check if the node has got terminal value or non-terminal.

- Terminal – node is a leaf, go further
- Non-terminal – go back to 5. point to create its children
- "$" value – node is a leaf, go further

8. Check if current node's value corresponds to the current character in given string.
    - It does – go to the next character in inputted string and next rule of CFG in array, then go 5.
    - It does not – do the Backtracking.

Backtracking:

1. Remove current node from the tree, go to its parent.
2. Check if there is any other production rule:
    - It is – go to 5. point in How it works:
    - It is not – prompt message that string does not belong to CFG.

Data structures

- CFG is stored as a dictionary of strings, each line from *.txt is a next key, value pair,
- PTree is a non-binary tree,
- Node keeps information about itself and its children (stored in python's tuple, sorted in order specified in file).

Module descriptions

(with algorithms at implementation level)

- Parser – parsing data from file and from input. Storing them in structures. Calling Parse_tree module to store manipulated data.
- PTree – module responsible for creation of non-binary tree together with its nodes. Stores data about root and children of each node.
- Checker – module responsible for checking validity of given data to the program.
- Node – module responsible for keeping data about a none in a tree.
- Tests – module with set of tests.
- Main – module with main function, calling Parser module.

Input/output description

Input:

*Give the file name: *file_path**
*Give the string to be checked: *string**

Output:

TUI representation of tree.
If string belongs to CFG the nodes representing the string are returned, otherwise the adequate message is being prompt.

Input file:

- File should contain CFG rules, each non-terminal character and its rules should be in one line in file.

- CFG should be written in such a form: ‘`S -> A|B|a`’, per line. User should pay attention to place blank spaces only nest to the arrow (‘->’).
- Each production rule for the same non-terminal character should be divided with “|”
- Non-terminal symbols cannot be repeated.
- CFG without “S” symbol would cause an error message.
- Each non-terminal symbol should have its rules of production.
- User should avoid using non-terminal characters with production rules pointing at self.

Input string:

- String can be “$” empty or “asdasda” not empty:
  - “$asda” – invalid
  - “as$ds” – invalid
  - “asddsa$” – invalid

Error Messages:

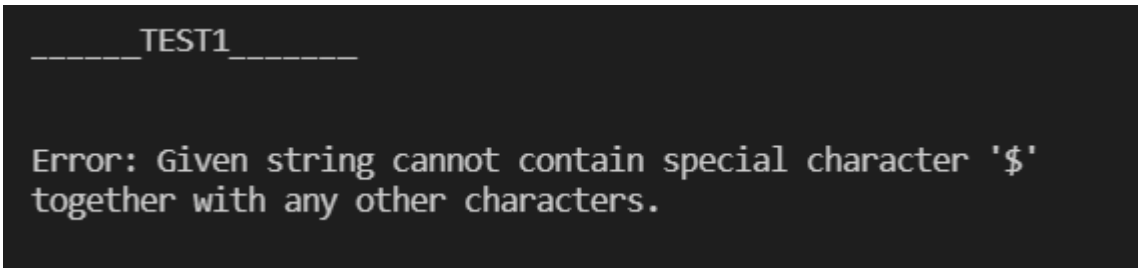
- Error: CFG should start with capital letter,
- Error: CFG should have non terminal (capital) letters at the beginning of each line,
- Error: CFG has to be written in a specific syntax, Non terminal symbol, space, arrow, space, (non)terminal symbols. Example: 'S -> a1|vBs|1,
- Error: Given string cannot contain special character together with any other characters,
- Error: Given string cannot contain capital letters,
- Error: spaces in file should be only next to an arrow ('->'), one to the left and one to the right,
- Error: There are some repetitions of non-terminal values in CFG,
- Error: There is at least one non-terminal value in CFG which is not defined in any rules of production,
- Error: CFG written in a way, that there is possibility to jump into infinite loop, while creating a tree,
- Error: Something went wrong.

Output Messages:

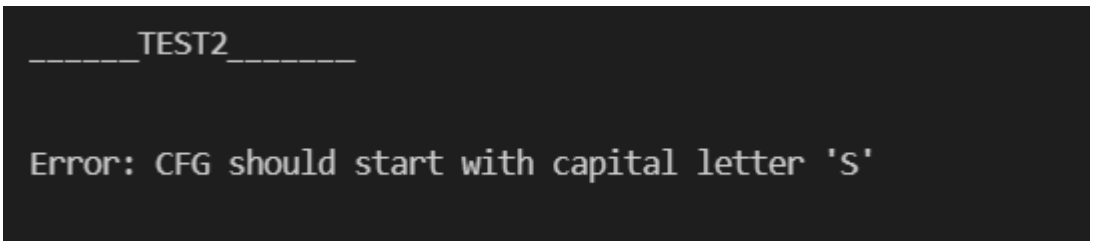
- ‘String’ not accepted by given CFG
- ‘String accepted by given CFG

IV. Functional test cases

1. S -> rXd|lZd
   X -> oa|ea
   Z -> a1
   String: read$
   CFG is valid, string contains special character '$'



2. A -> A|B|a
   B -> a|aa|aaa
   No starting symbol "S"



3. S -> r$Xd|lXd
   X -> oa|Z
   Z -> B|C
   B -> $
   C -> $|FF
   F -> b|e|$E
   E -> a|C
   String: lead
   CFG is complex, but valid, string belongs to CFG

```
______TEST3_______

====================
Non-terminal Node: S
with children: r$Xd

====================

Child for which we need backtracking: r
Backtracking...


Deleting node: S

====================
Non-terminal Node: S
with children: lXd

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: oa

====================

Child for which we need backtracking: o
Backtracking...


Deleting node: X

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: B

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: B
```

```
Non-terminal Node: B
with children: $

====================

Backtracking...


Deleting node: B


Yet another backtracking...


Deleting node: Z

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z
```

```
Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: $

====================

Backtracking...


Deleting node: C

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z
```

```
Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Non-terminal Node: F
with children: b

====================

Child for which we need backtracking: b
Backtracking...


Deleting node: F

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Non-terminal Node: F
with children: e

====================
```

```
 ====================
 Non-terminal Node: S
 with children: lXd

 Terminal Node: l

 Non-terminal Node: X
 with children: Z

 Non-terminal Node: Z
 with children: C

 Non-terminal Node: C
 with children: FF

 Non-terminal Node: F
 with children: e

 Terminal Node: e

 ====================
 ====================
 Non-terminal Node: S
 with children: lXd

 Terminal Node: l

 Non-terminal Node: X
 with children: Z

 Non-terminal Node: Z
 with children: C

 Non-terminal Node: C
 with children: FF
```

```
Non-terminal Node: F
with children: e

Terminal Node: e

Non-terminal Node: F
with children: b

====================

Child for which we need backtracking: e
Backtracking...


Deleting node: F

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF
```

```
Terminal Node: e

Non-terminal Node: F
with children: e

====================

Child for which we need backtracking: e
Backtracking...


Deleting node: F

====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Terminal Node: e

Non-terminal Node: F
with children: $E

====================
====================
Non-terminal Node: S
with children: lXd
```

```
Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Terminal Node: e

Non-terminal Node: F
with children: $E

Non-terminal Node: E
with children: a

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Terminal Node: e

Non-terminal Node: F
with children: $E
```

```
Non-terminal Node: E
with children: a

Terminal Node: a

====================
====================
Non-terminal Node: S
with children: lXd

Terminal Node: l

Non-terminal Node: X
with children: Z

Non-terminal Node: Z
with children: C

Non-terminal Node: C
with children: FF

Terminal Node: e

Non-terminal Node: F
with children: $E

Non-terminal Node: E
with children: a

Terminal Node: a

Terminal Node: d

====================
```

```
Terminal Node: l

Terminal Node: e

Terminal Node: a

Terminal Node: d
```

4. S -> A|aB
   A -> b|A
   B -> d
   String: ads
   Non terminal ‘A’, with the production pointing at ‘A’

```
______TEST4_______


Error: CFG written in a way, that there is possibility
to jump into infinite loop, while creating a tree
```

5. S -> X
   X -> A|a
   String: cxxc
   Non terminal ‘A’ has not got specified production rules

```
______TEST5_______


Error: There is at least one non-terminal value in CFG
which has not its rules of production defined
```

6. S -> Abb
   A -> aab
   String: aabbb
   Valid CFG with valid, belonging string

```
______TEST6_______

====================
Non-terminal Node: S
with children: Abb

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

Terminal Node: a

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

Terminal Node: a
```

```
Terminal Node: a

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

Terminal Node: a

Terminal Node: a

Terminal Node: b

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

Terminal Node: a

Terminal Node: a

Terminal Node: b

Terminal Node: b
```

```
====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

Terminal Node: a

Terminal Node: a

Terminal Node: b

Terminal Node: b

Terminal Node: b

====================
Terminal Node: a

Terminal Node: a

Terminal Node: b

Terminal Node: b

Terminal Node: b
```

7. S -> Abb
   A -> aab
   String: bbaa
   Valid CFG with valid, not belonging string

```
______TEST7_______

====================
Non-terminal Node: S
with children: Abb

====================
====================
Non-terminal Node: S
with children: Abb

Non-terminal Node: A
with children: aab

====================

Child for which we need backtracking: a
Backtracking...


Deleting node: A


Yet another backtracking...


Deleting node: S


Yet another backtracking...
```

```
====================
Terminal Node: b

====================
====================
Terminal Node: b

Terminal Node: b

====================

Error: Program could not process string in such CFG
```