PRACTICAL No. 4

Name : Shantanu Mane

Roll No. : E-63

Topic: Parsing

Platform: Windows or Linux

Language to be used: Python or Java (based on the companies targeted for placement)

Aim: (A) Write a program to validate a natural language sentence. Design a natural language grammar, compute and input the LL (1) table. Validate if the given sentence is valid or not based on the grammar.

Input: NLP grammar and LL (1) parsing table (from file)
Implementation: String parsing rules
Output: Each step-in string parsing and whether the input string is valid or invalid.

CODE :

```
import pandas as pd


def util(ll1):
    tab = pd.DataFrame(
        ll1,
        columns=[
            "championship",
            "ball",
            "toss",
            "is",
            "want",
            "won",
            "Played",
            "me",
            "I",
            "you",
            "India",
            "Australia",
            "Steve",
            "John",
            "the",
            "a",
            "an",
        ],
    )
    tab["Nonterm"] = ["S", "NP", "VP", "N", "V", "P", "PN", "D"]
    tab.set_index("Nonterm", inplace=True)
    return tab


def validator(input):
```

```
from beautifultable import BeautifulTable

ll1 = [
    [
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
        "NP VP",
    ],
    [
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "P",
        "P",
        "P",
        "PN",
        "PN",
        "PN",
        "PN",
        "D N",
        "D N",
        "D N",
    ],
    [
        "",
        "",
        "",
        "",
        "V NP",
        "V NP",
        "V NP",
        "V NP",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
        "",
```

```
            "",
            "",
            "",
        ],
        [
            "championship",
            "ball",
            "toss",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
        ],
        [
            "",
            "",
            "",
            "",
            "is",
            "want",
            "won",
            "played",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
        ],
        ["", "", "", "", "", "", "", "me", "I", "you", "", "", "", "",
"", "", ""],
        [
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "",
            "India",
```

```
            "Australia",
            "Steve",
            "John",
            "",
            "",
            "",
        ],
        ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "the",
"a", "an"],
    ]
    tab = util(ll1)
    table = BeautifulTable()
    table.column_headers = ["Buffer", "Stack"]
    buffer = input.split(" ")
    buffer.reverse()
    stack = ["S"]
    table.append_row([buffer.copy(), stack.copy()])
    while buffer != [] and stack != []:
        index = stack.pop(0)
        key = buffer[-1]
        if key not in tab.columns:
            print("Invalid input")
            table.append_row([buffer.copy(), stack.copy()])
            print(table)
            return

        rule = tab.loc[index][key].split(" ")

        if "" in rule:
            print("Invalid input")
            table.append_row([buffer.copy(), stack.copy()])
            print(table)
            return

        stack = rule + stack
        table.append_row([buffer.copy(), stack.copy()])

        if key in rule:
            buffer.remove(key)
            stack.remove(key)
            table.append_row([buffer.copy(), stack.copy()])
    print(table)
    print("Valid input")


input = "India won the championship"
validator(input)
input = "championship India won"
validator(input)
```

OUTPUT :

```
+-----------------------------------------+------------------+
|                 Buffer                  |      Stack       |
+-----------------------------------------+------------------+
| ['championship', 'the', 'won', 'India'] |      ['S']       |
```

```
+-----------------------------------------+------------------+
| ['championship', 'the', 'won', 'India'] |   ['NP', 'VP']   |
+-----------------------------------------+------------------+
| ['championship', 'the', 'won', 'India'] |   ['PN', 'VP']   |
+-----------------------------------------+------------------+
| ['championship', 'the', 'won', 'India'] | ['India', 'VP']  |
+-----------------------------------------+------------------+
|     ['championship', 'the', 'won']      |      ['VP']      |
+-----------------------------------------+------------------+
|     ['championship', 'the', 'won']      |   ['V', 'NP']    |
+-----------------------------------------+------------------+
|     ['championship', 'the', 'won']      |  ['won', 'NP']   |
+-----------------------------------------+------------------+
|         ['championship', 'the']         |      ['NP']      |
+-----------------------------------------+------------------+
|         ['championship', 'the']         |    ['D', 'N']    |
+-----------------------------------------+------------------+
|         ['championship', 'the']         |   ['the', 'N']   |
+-----------------------------------------+------------------+
|            ['championship']             |      ['N']       |
+-----------------------------------------+------------------+
|            ['championship']             | ['championship'] |
+-----------------------------------------+------------------+
|                   []                    |        []        |
+-----------------------------------------+------------------+
Valid input
Invalid input
+---------------------------------+-------+
|              Buffer             | Stack |
+---------------------------------+-------+
| ['won', 'India', 'championship'] | ['S'] |
+---------------------------------+-------+
| ['won', 'India', 'championship'] |  []   |
+---------------------------------+-------+
```

(B) Use Virtual Lab on LL1 parser to validate the string and verify your string validation using simulation.

Link for Virtual Lab: http://vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/system_deligators/labs/exp2/index.php

Output: Validation from Virtual lab simulator

Details:

PART A:

- Construct and consider a natural language grammar that can validate an English sentence.
- Solve the NLP grammar by hand for LL(1) parser and create parsing table
- Input the above parsing table and grammar using a file.
- Write program for performing string validation

PART B:

- Go to Virtual lab: Go through all the tabs, paste screen shots for all steps (including tests), validate your string parsing with the simulator (screen shot expected).

Not secure | vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/system_deligators/labs/exp2/pretest.php

IoT Based Fire Alar... Control Lights with... https://www.astagu... (DOC) Fall of Indian... Heart Disease with...

○ code generation

4) E = E + T is correct grammar to parse the string without back tracing?
○ Yes
◉ No

5) Grammar is ambiguous if _____
○ it has 2 parse trees+
○ it has two left most derivations
○ it has two right most derivations
◉ all of the above

6) How to generate parse table?
◉ Using First and Follow
○ Using RSA algo

7) The concept of grammar is much used in this part of the compiler
○ lexical analysis
◉ parser
○ code generation
○ code optimization

8) Examine the given input string is syntactically correct or not id ** id
○ Correct
◉ Incorrect

Evaluate

1) Correct

2) Correct

3) Correct

4) Correct

5) Correct

6) Correct

7) Correct

8) Correct

Lab contributed by K S School of Bussiness Management, Gujarat University

Not secure | vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/system_deligators/labs/exp2/posttest.php

IoT Based Fire Alar... Control Lights with... https://www.astagu... (DOC) Fall of Indian... Heart Disease with...

Simulation
Post Test
References

1) Once you land on the simulator page what is the default condition grammar for LL(1) parser?
◉ Left recursive eliminated grammar
○ Left recursive grammar
○ Right recursive grammar
○ Any grammar

2) What result do you expect when top down parsing will perform using right recursive grammar?
○ Infinite loop
◉ Top down parsing with Back tracking
○ Parse the given input
○ Syntax analysis

3) What importance of parser table to parse the given input string?
○ Top down parsing with back tracking
○ Bottom up parsing
○ A formal grammar that contains left recursion cannot be parsed by a LL(k)-parser
◉ Automatic Predictive top down parsing of the input string without backtracking

4) What do you observe when left recursive grammar is used without rewriting?
○ A formal grammar that contains left recursion can be parsed by a LL(k)-parser
○ Top down parsing with Back tracking
◉ It leads them into infinite recursion
○ Bottom up parsing

5) What is the use of first and follow algorithm?
○ LL1 parsing of the given input string
○ To generate left recursive eliminated grammar
○ To generate left recursive grammar
◉ predictive parse table construction for LL1 parsing

6) Generate LL(1) parsing for the grammar given in Task 2 for given input string (L04) (L05) id + id ** (id + id)
id + * id + (id * id)
7) analyze
id*+id

Evaluate

1) Correct

2) Correct

3) Correct

4) Correct

5) Correct

LL(1) Parser Visualization

Write your own context-free grammar and see an LL(1) parser in action!

Written by Zak Kincaid and Shaowei Zhu based on
http://jsmachines.sourceforge.net/machines/ll1.html

1. Write your LL(1) grammar (empty string '' represents ε):

```
S ::= NP VP
NP ::= P
NP ::= PN
NP ::= D N
VP ::= V NP
N ::= championship
N ::= ball
N ::= toss
V ::= is
V ::= want
V ::= won
V ::= played
P ::= me
P ::= I
P ::= you
PN ::= India
PN ::= Australia
PN ::= Steve
PN ::= John
D ::= the
D ::= a
D ::= an
```

Generate tables

Valid LL(1) Grammars

For any production S -> A | B, it must be the case that:

- For no terminal t could A and B derive strings beginning with t
- At most one of A and B can derive the empty string
- If B can derive the empty string, then A does not derive any string beginning with a terminal in Follow(A)

Formatting Instructions

- The non-terminal on the left-hand-side of the first rule is the start non-terminal
- Write each production rule in a separate line (see example to the left)
- Separate each token using whitespace
- $ is reserved as the end-of-input symbol, and S is reserved as an artificial start symbol. The grammar is automatically augmented with the rule S ::= *start* $

Debugging

- More information about the parser construction is printed on the console
- The source code follows the pseudocode in lecture. In particular, see `computeNullable`, `computeFirst`, `computeFollow`, and `computeLLTables`

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	×	me, I, you, India, Australia, Steve, John, the, a, an	$
NP	×	me, I, you, India, Australia, Steve, John, the, a, an	is, want, won, played, $
VP	×	is, want, won, played	$
N	×	championship, ball, toss	is, want, won, played, $
V	×	is, want, won, played	me, I, you, India, Australia, Steve, John, the, a, an
P	×	me, I, you	is, want, won, played, $
PN	×	India, Australia, Steve, John	is, want, won, played, $
D	×	the, a, an	championship, ball, toss

	$	championship	ball	toss	is	want	won	played	me	I	you	India	Australia	Steve	John	the	a	an
S									S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $	S ::= NP VP S ::= S $
NP									NP ::= P	NP ::= P	NP ::= P	NP ::= PN	NP ::= PN	NP ::= PN	NP ::= PN	NP ::= D N	NP ::= D N	NP ::= D N
VP					VP ::= V NP	VP ::= V NP	VP ::= V NP	VP ::= V NP										
N		N ::= championship	N ::= ball	N ::= toss														
V					V ::= is	V ::= want	V ::= won	V ::= played										
P									P ::= me	P ::= I	P ::= you							
PN												PN ::= India	PN ::= Australia	PN ::= Steve	PN ::= John			
D																D ::= the	D ::= a	D ::= an

3. Parsing

Token stream separated by spaces: India won the championship

Start/Reset | Step Forward

Stack

$

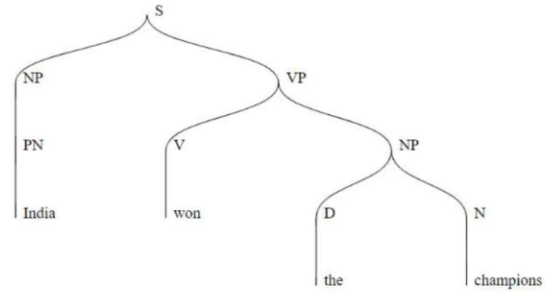
Remaining Input

$

Rule

Match championship

Partial Parse Tree



	Championship	ball	toss	is	want	won	Played	me	I	you	2nd	Aus	Steve	John	the	a	an
S				S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	S→NP VP	
NP								NP→P	NP→P	NP→P	NP→PN	NP→PN	NP→PN	NP→PN	NP→AN	NP→AN	NP→AN
VP				VP→V NP	VP→VNP	VP→V NP	VP→V NP										
N	N→Champ		N→ball	N→toss													
V				V→is	V→want	V→won	V→played										
P								P→me	P→I	P→you							
PN											PN→2nd	PN→Aus	PN→Steve PN	PN→John			
D															D→the	D→a	D→an