**Name: Bhavesh Kewalramani**

**Roll No.: A-25**

**Practical No. 4**

**Theory**

**LL(1) Parsing:**   
Here the 1st **L** represents that the scanning of the Input will be done from Left to Right manner and the second **L** shows that in this parsing technique we are going to use Left most Derivation Tree. And finally, the **1** represents the number of look-ahead, which means how many symbols are you going to see when you want to make a decision.

**Algorithm to construct LL(1) Parsing Table:**

**Step 1:**First check for left recursion in the grammar, if there is left recursion in the grammar remove that and go to step 2.

**Step 2:**Calculate First() and Follow() for all non-terminals.

1. **First():** If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.
2. Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

**Step 3:**For each production A –> α. (A tends to alpha)

1. Find First(α) and for each terminal in First(α), make entry A –> α in the table.
2. If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.
3. If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α in the table for the $.  
   To construct the parsing table, we have two functions:

In the table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the **Null Productions** of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

**Practicals**

**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 based on the grammar or not.

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

using simulation.

**Program:**

from collections import OrderedDict

import re

# ` reprsents EPSILON

def getGrammar():

terminal=[]

nonterminal=[]

start=""

rule=dict()

flag=0

print("Enter the Production Rules : ")

while(True):

inp=input("===> ")

if(inp==""):

break

s1,s2=inp.split("~")

if flag==0:

start=s1

flag=1

rule[s1]=[]

s2=list(s2.split("/"))

for i in s2:

rule[s1].append(i)

r=[]

k=rule.keys()

for i in rule.values():

for j in i:

for a in list(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):

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(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):

print("Given String is : ", string)

string = list(string.split(' '))

string.append('$')

stk = ['$', start]

print("Input Stack")

print(string[::-1] , "\t\t\t\t\t", stk[::-1])

while not len(stk) == 0:

top = stk[-1]

stk.pop()

if string[0] == '$' and top == '$' and len(stk) == 0:

print("\*\*\*\*\* String Accepted \*\*\*\*\*")

elif (string[0] == '$' and len(stk) != 0) or (string[0] != '$' and len(stk) == 0):

print("\*\*\*\*\* String Rejected \*\*\*\*\*")

elif top == string[0]:

string = string[1:]

else:

for i in t[top][string[0]]:

l = list(i.split(' '))

for j in l[::-1]:

stk.append(j)

print(stk[::-1] , "\t\t\t\t\t", stk[::-1])

if \_\_name\_\_=="\_\_main\_\_":

d,start,r,t=getGrammar()

print("===== FIRST =====")

for i in d.keys():

print("first(", i, ") : ", Calculate\_First([i],d,r))

print("===== FOLLOW =====")

for i in d.keys():

print("follow(", i, ") : ", Calculate\_Follow(i,d,r,start))

print("===== Parsing Table =====")

for i in d.keys():

for rule in d[i]:

f = Calculate\_First(list(rule.split(' ')),d,r)

if '`' in f:

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 f:

t[i][j].add(rule)

print("\t\t", end=" ")

for i in r:

print(i, end="\t\t\t\t")

print()

print("====== String Parsing =====")

for j in r:

print("Terminal : ", j)

for i in d.keys():

if len(t[i][j]) != 0:

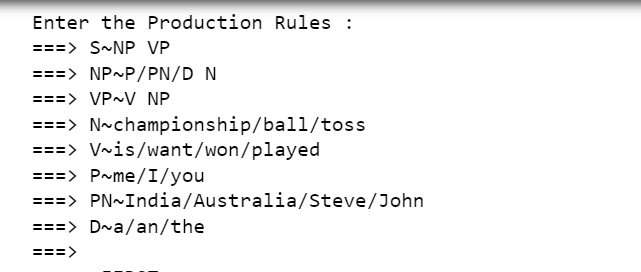
print(i, "->", t[i][j])

print()

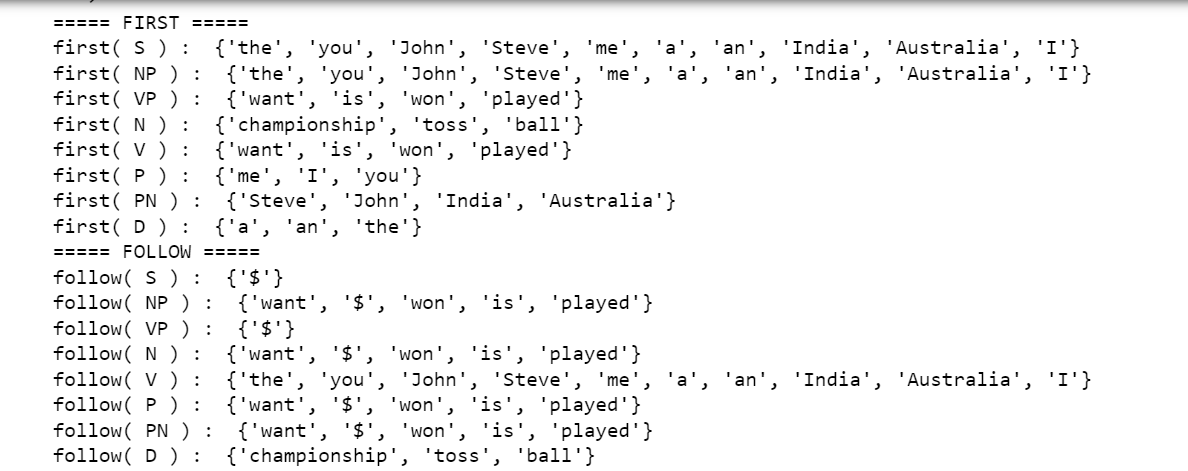
string = "India won the championship"

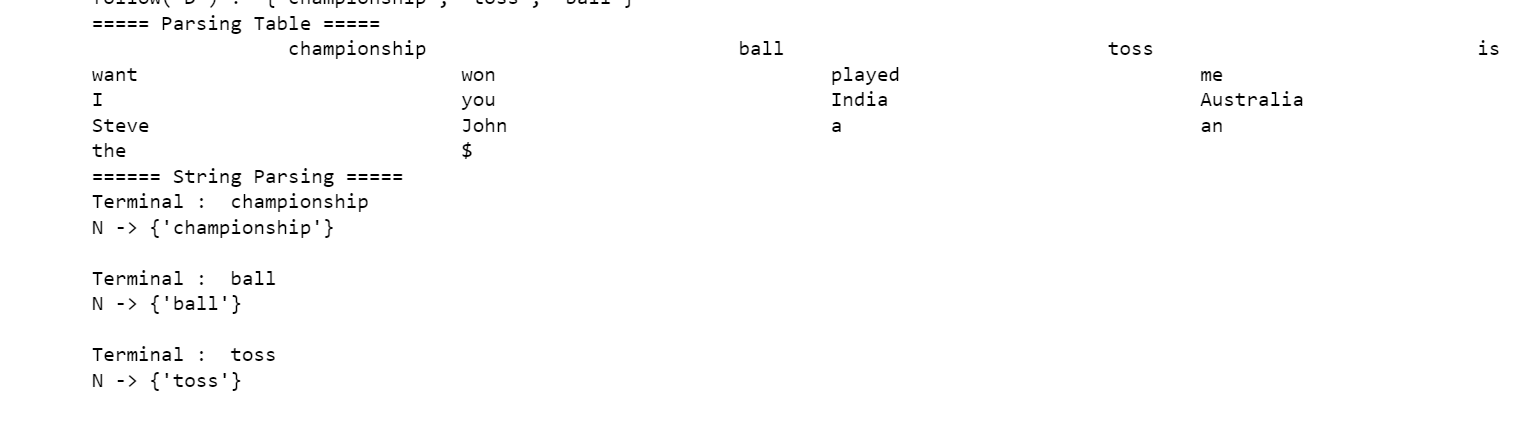
parseString(string,t)

**Input:**

****

**Output:**

****

****

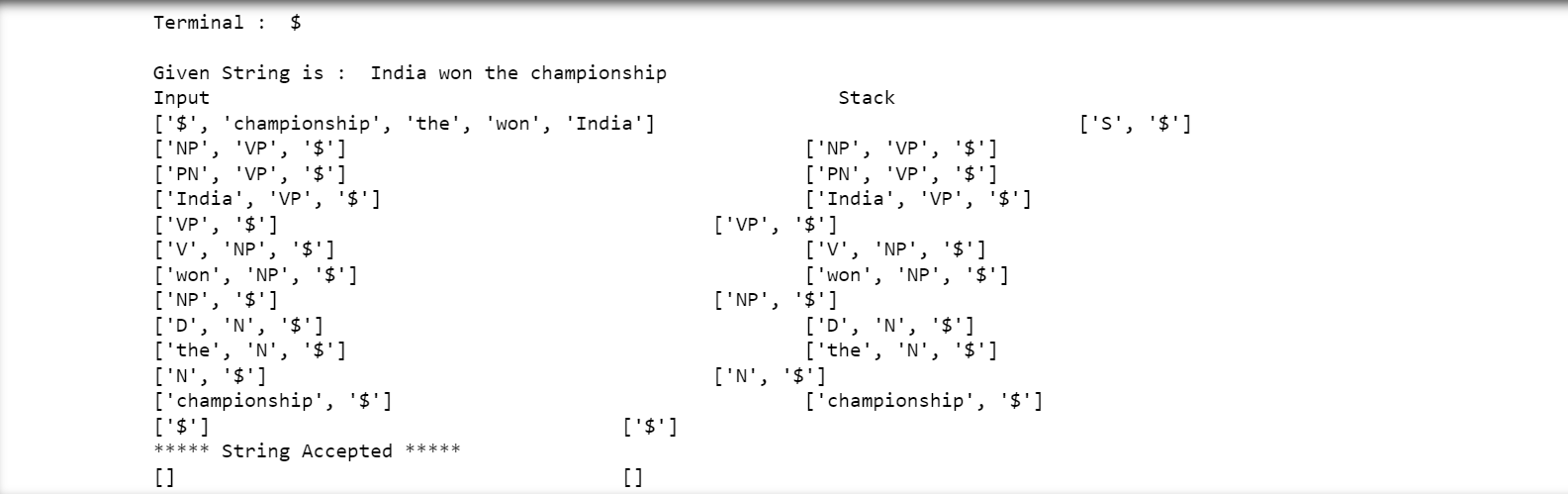
****

****

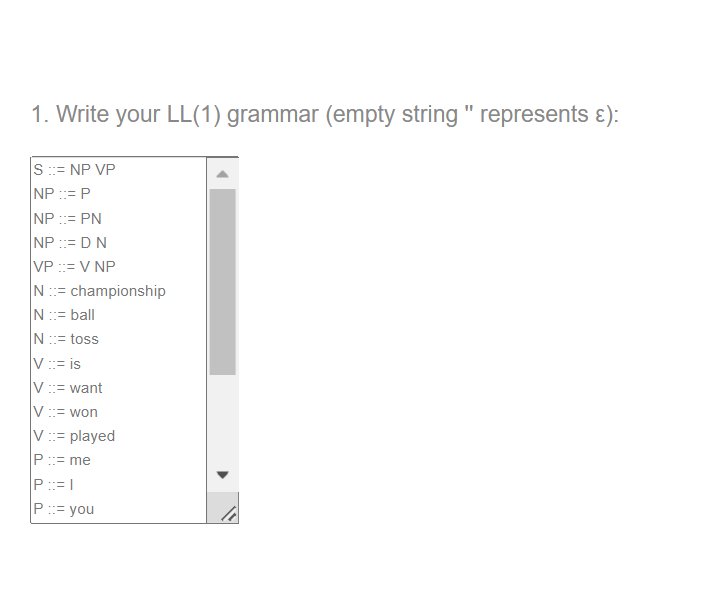
****

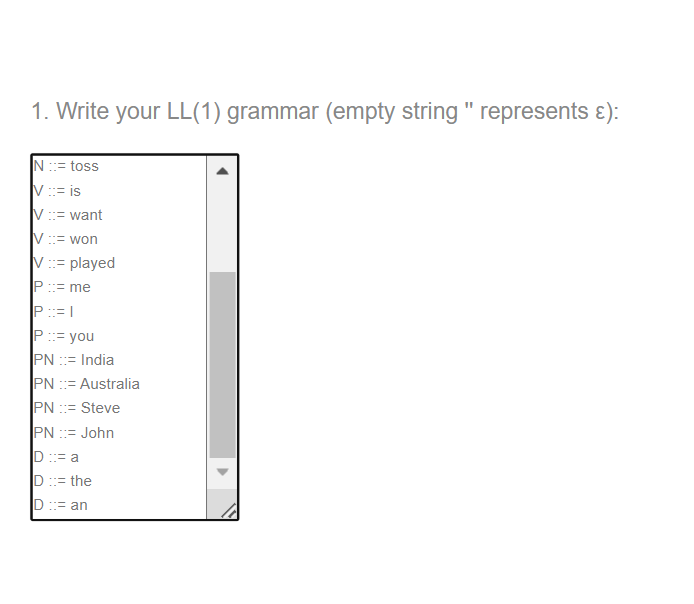
****

****

****

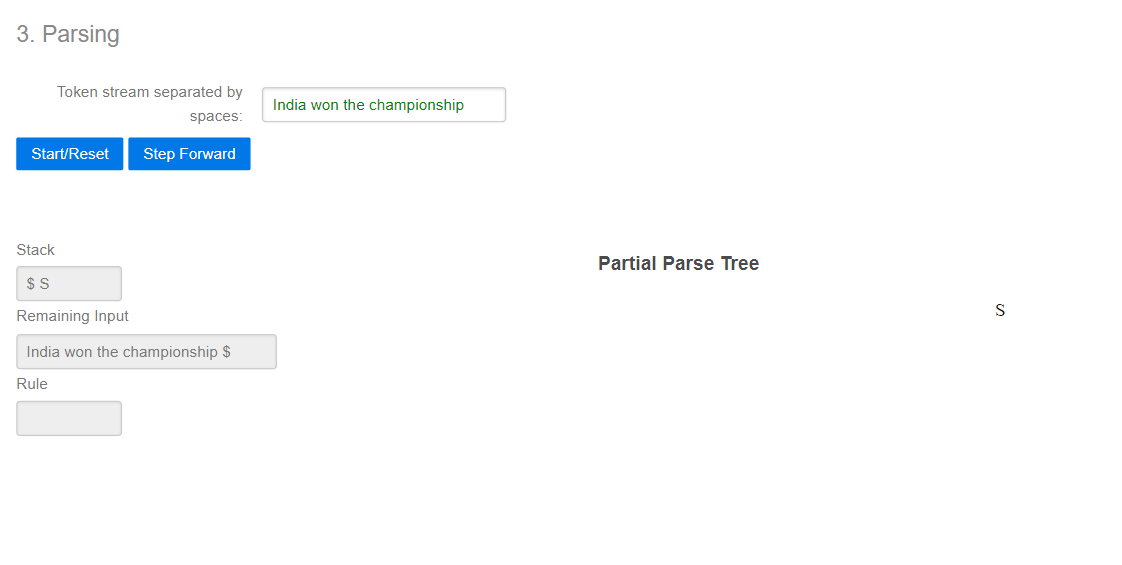
**Virtual Lab Simulation:**

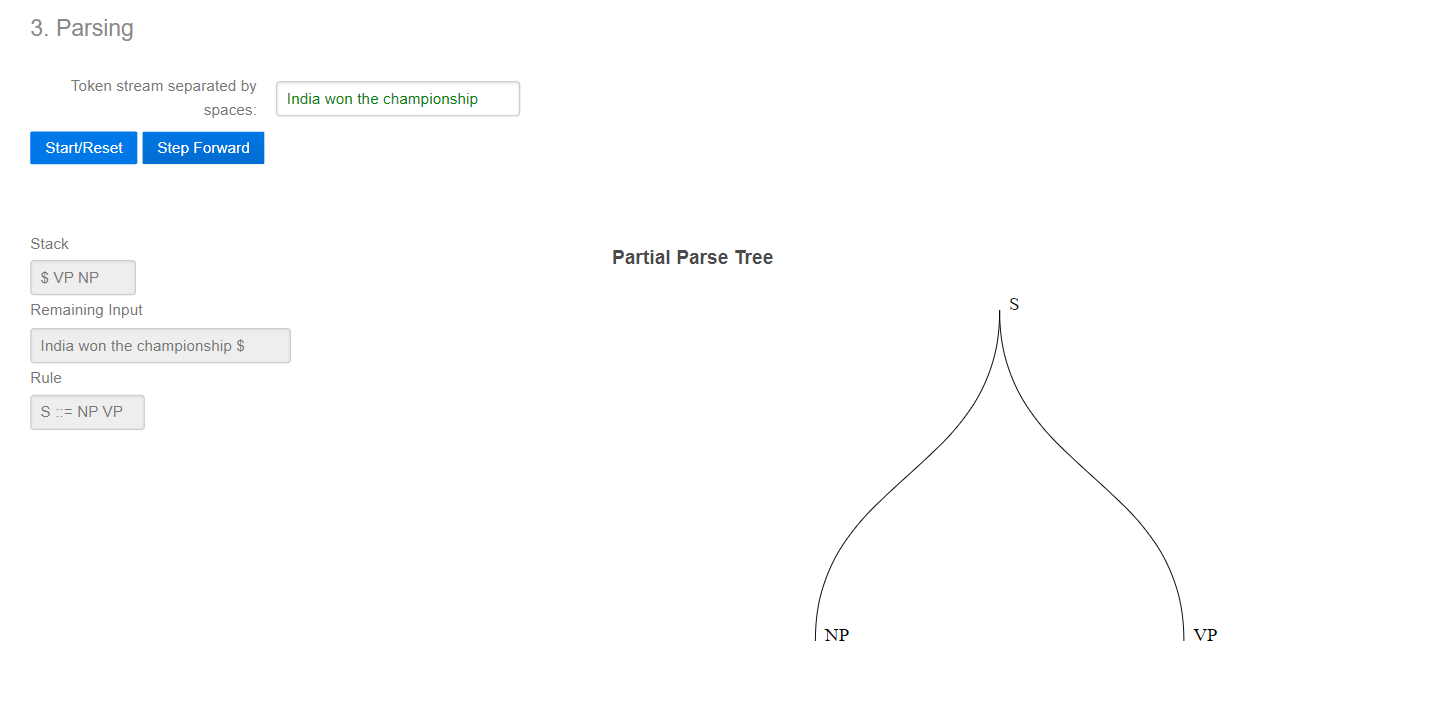
****

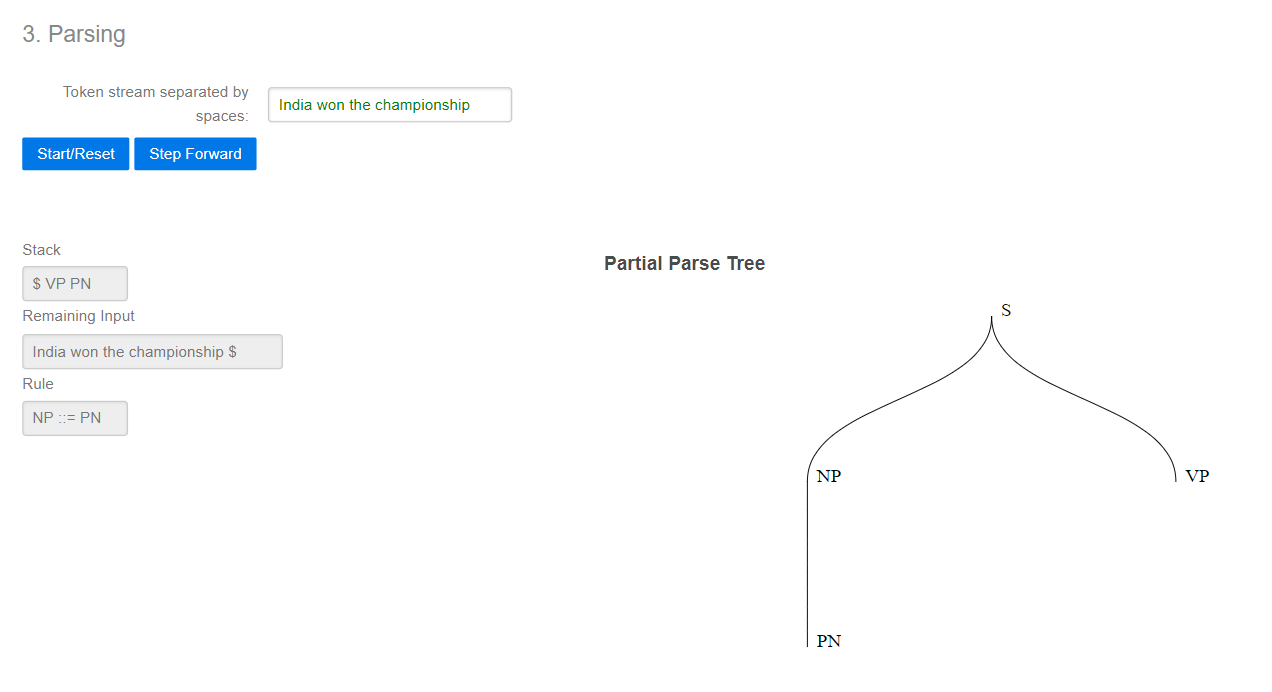
****

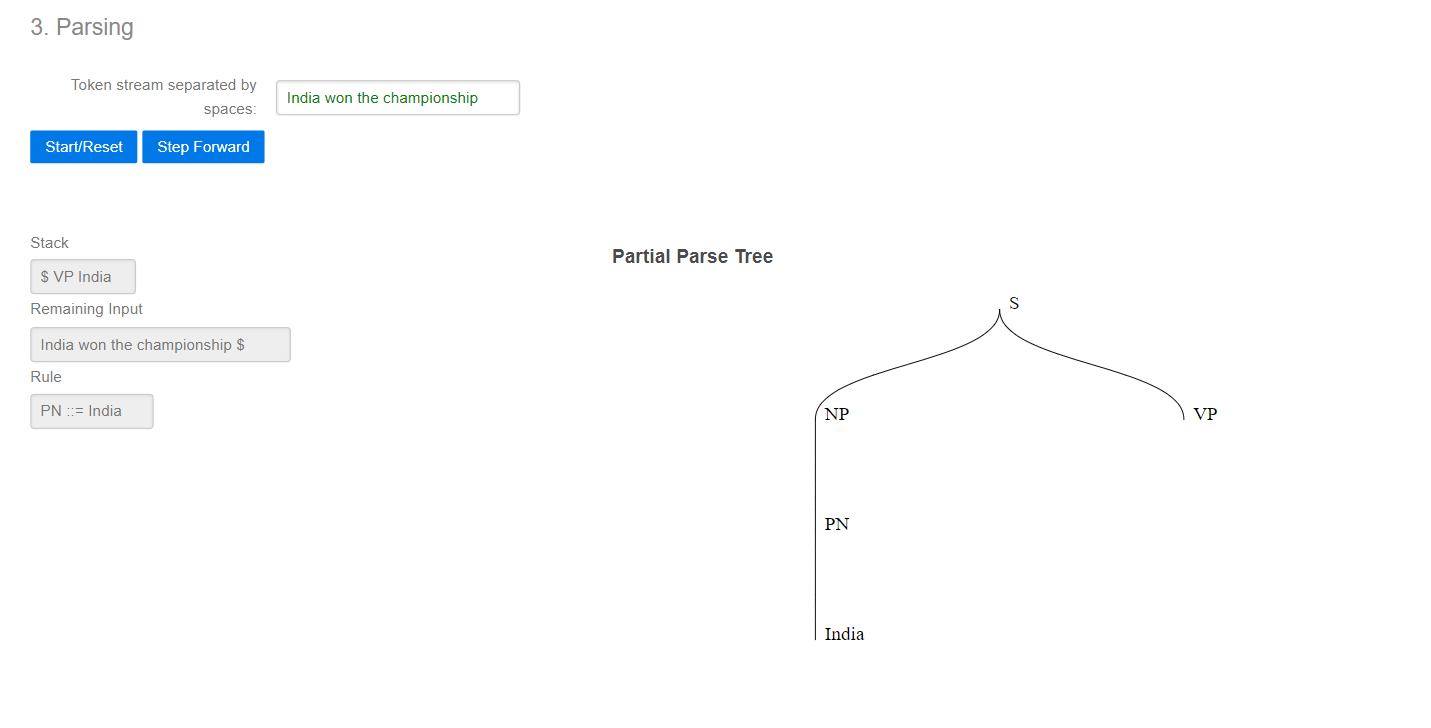
****

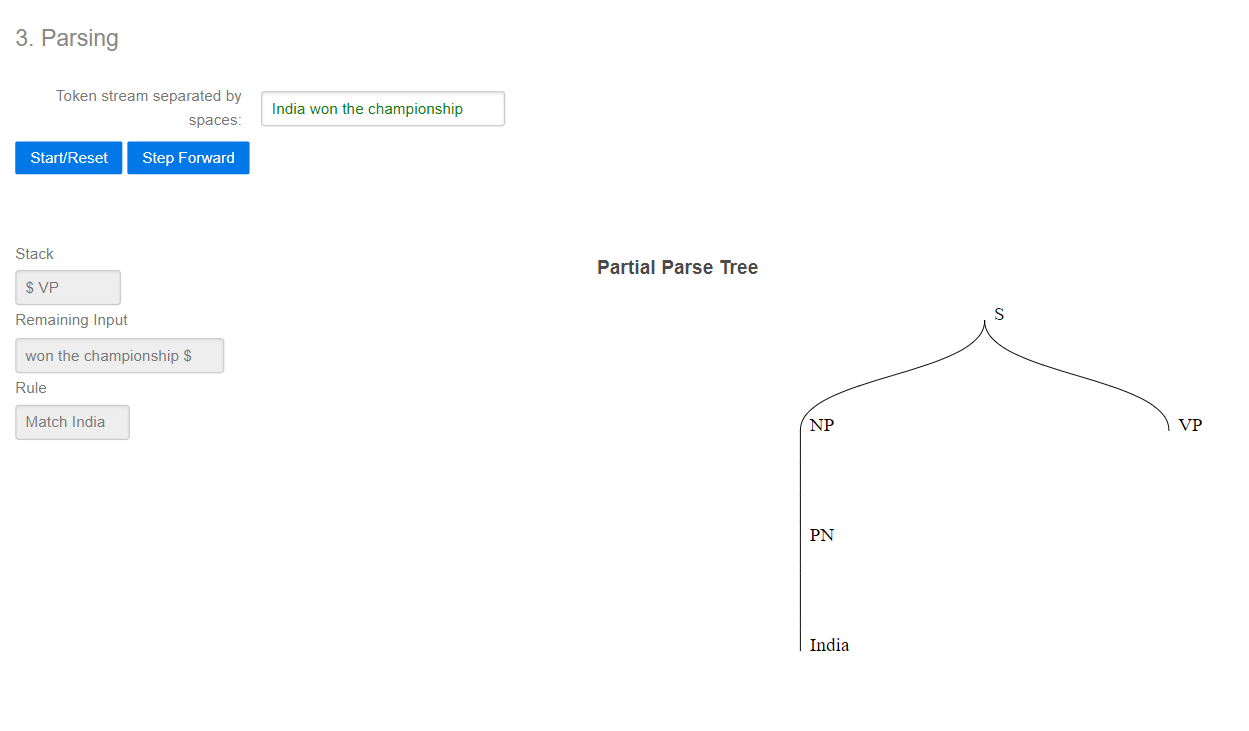
****

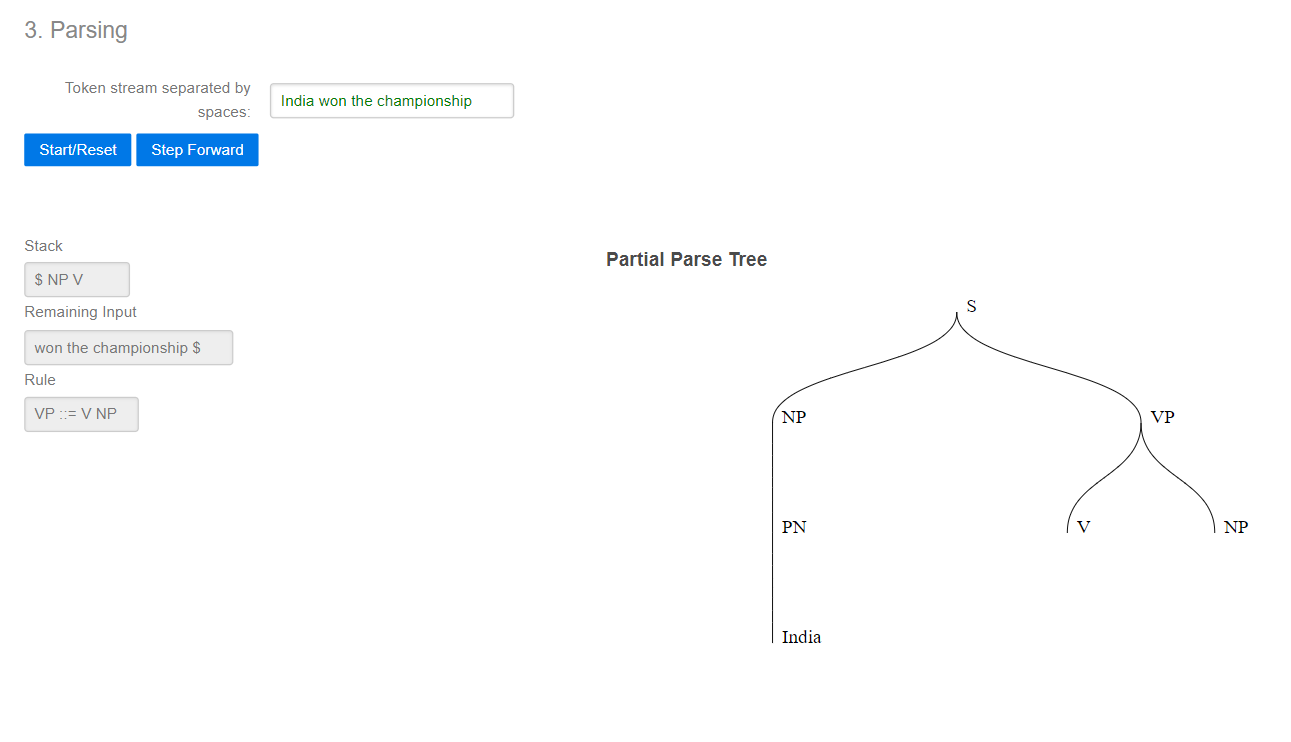
****

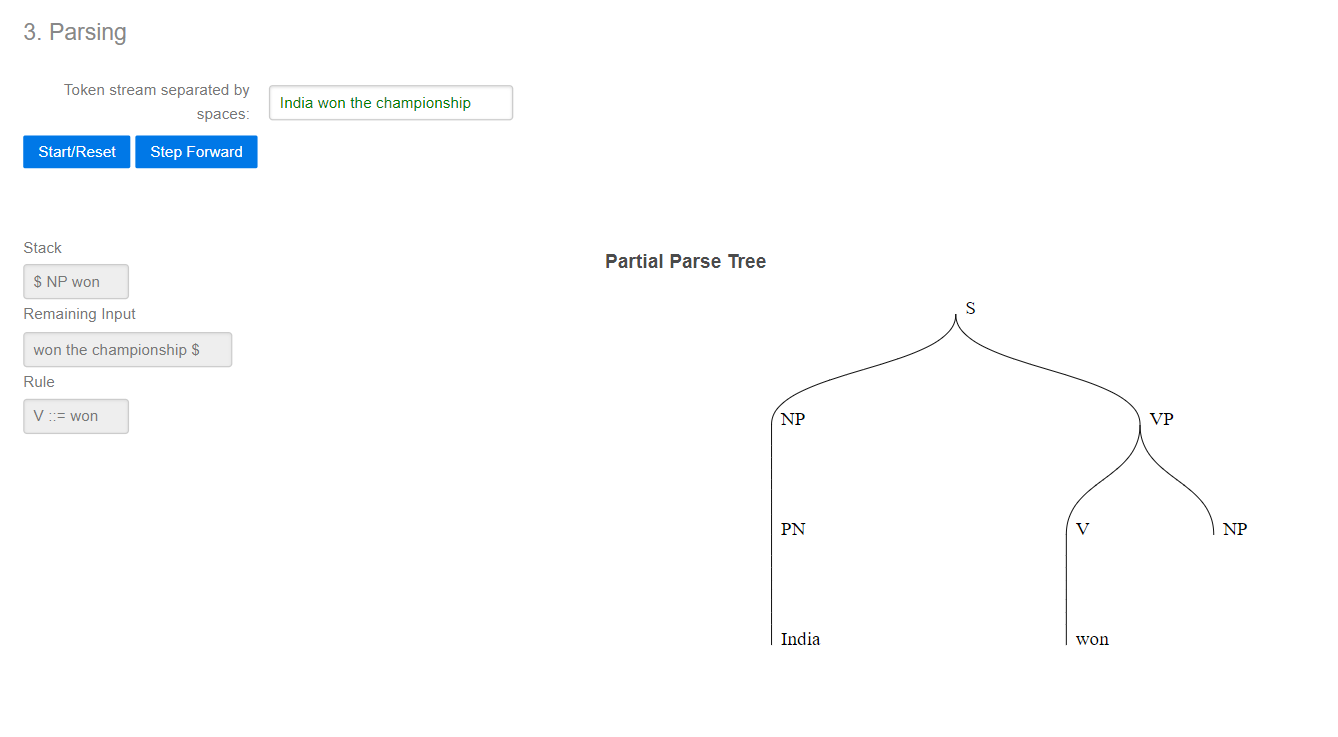
****

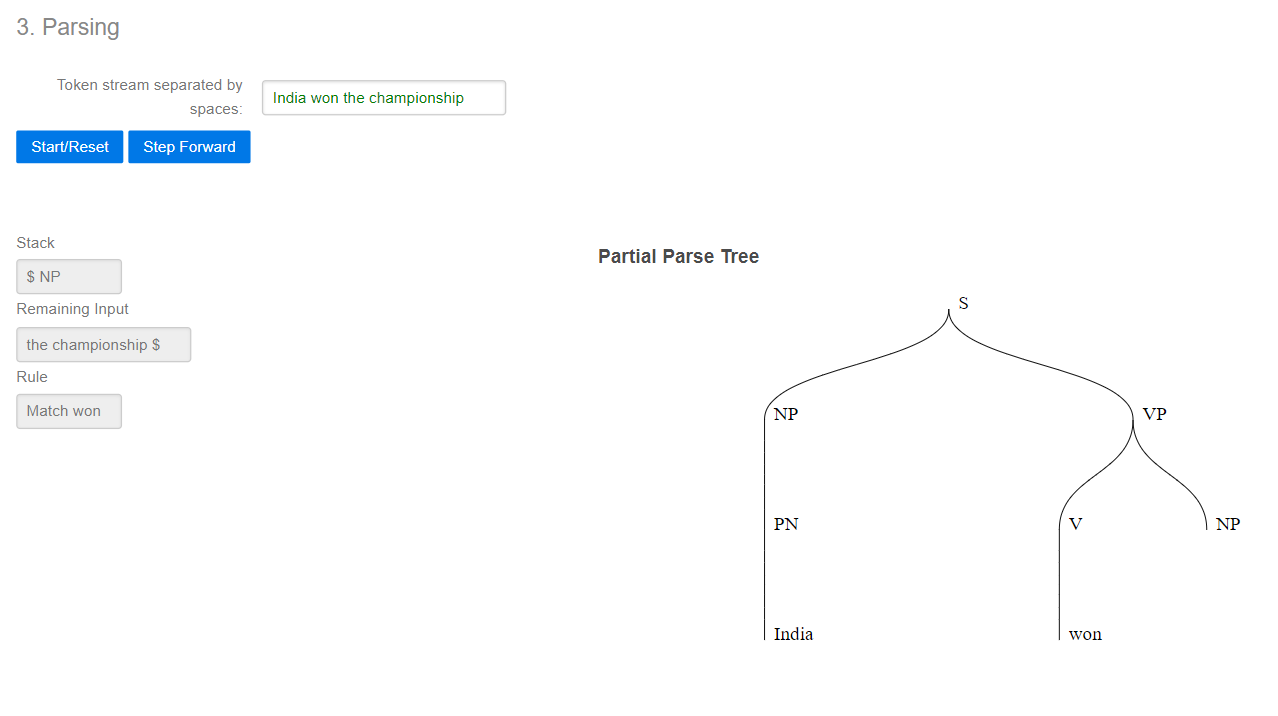
****

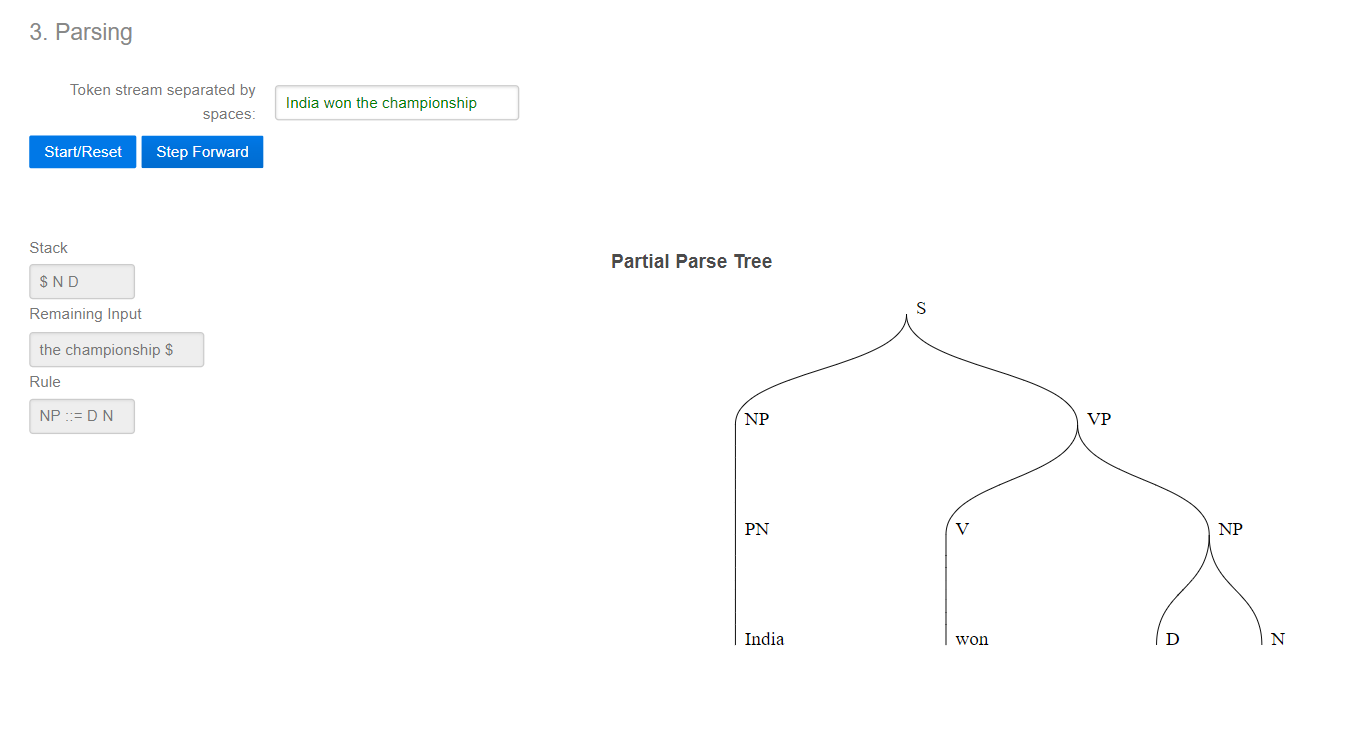
****

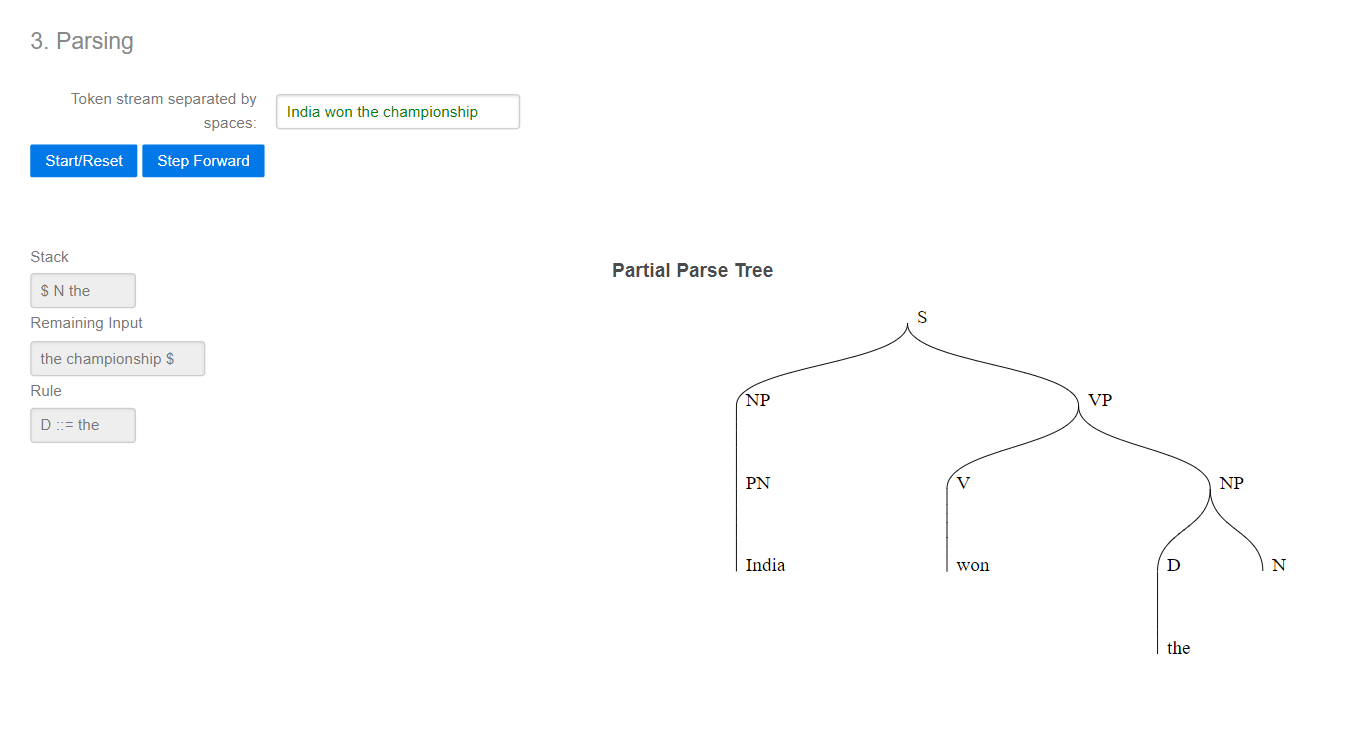
****

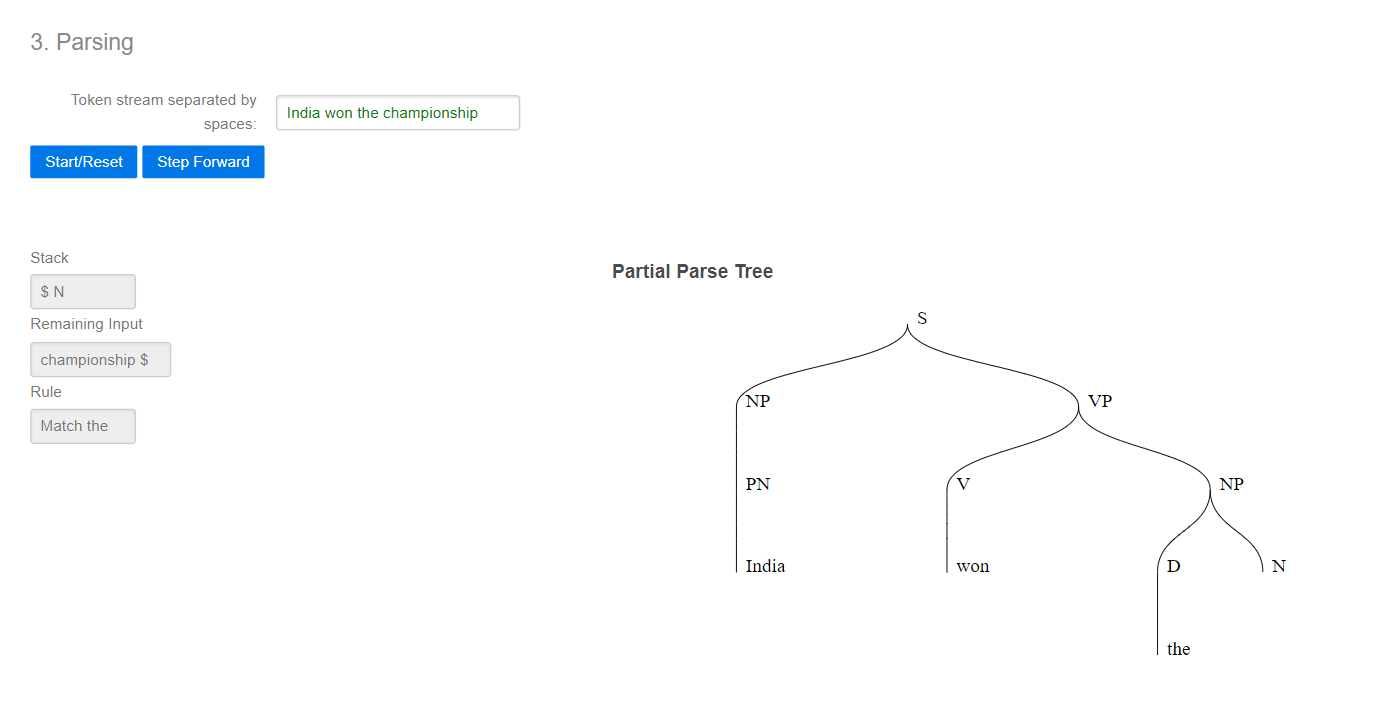
****

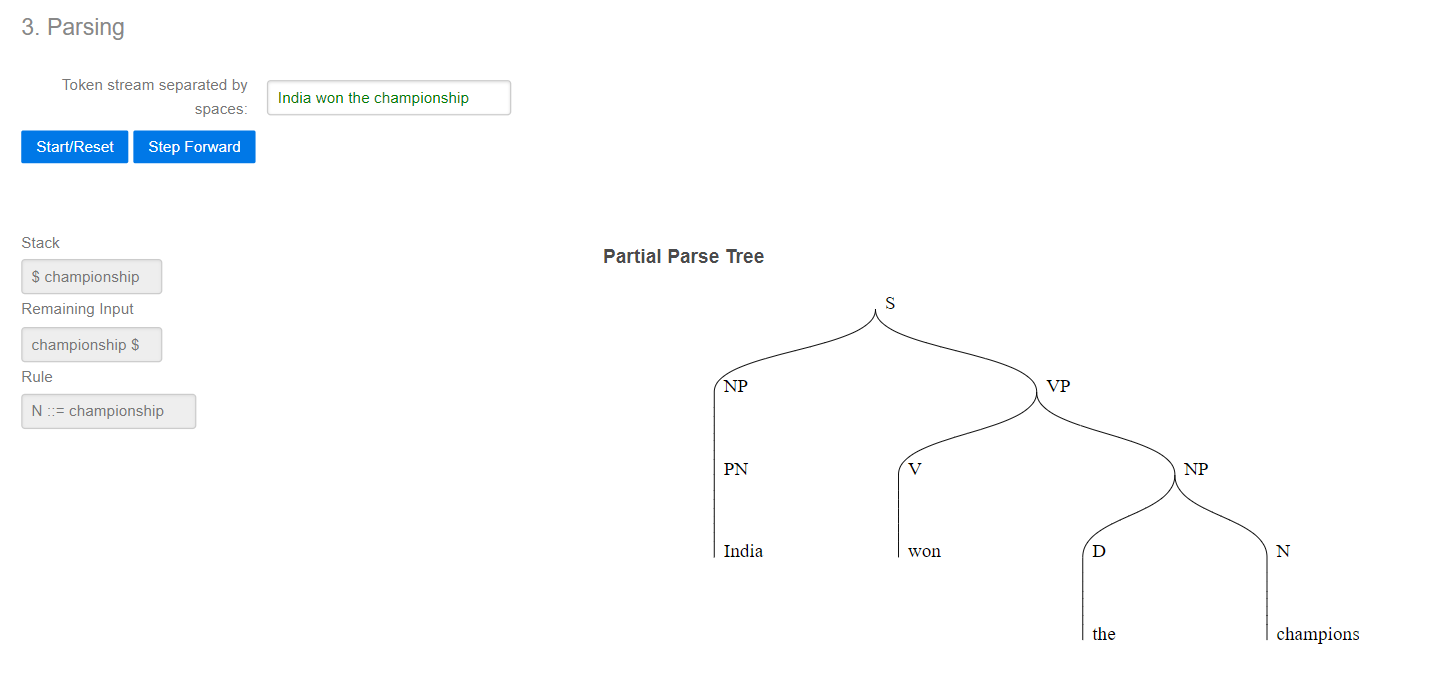
****

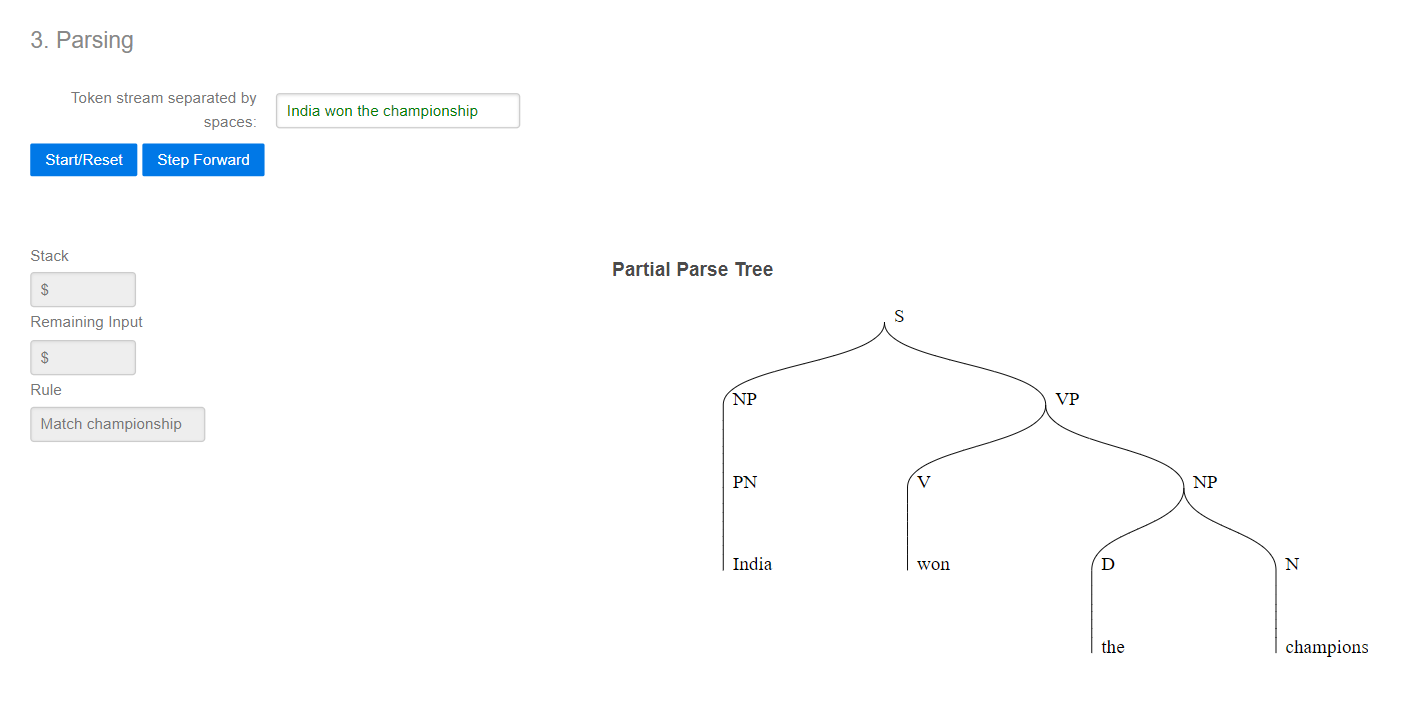
****

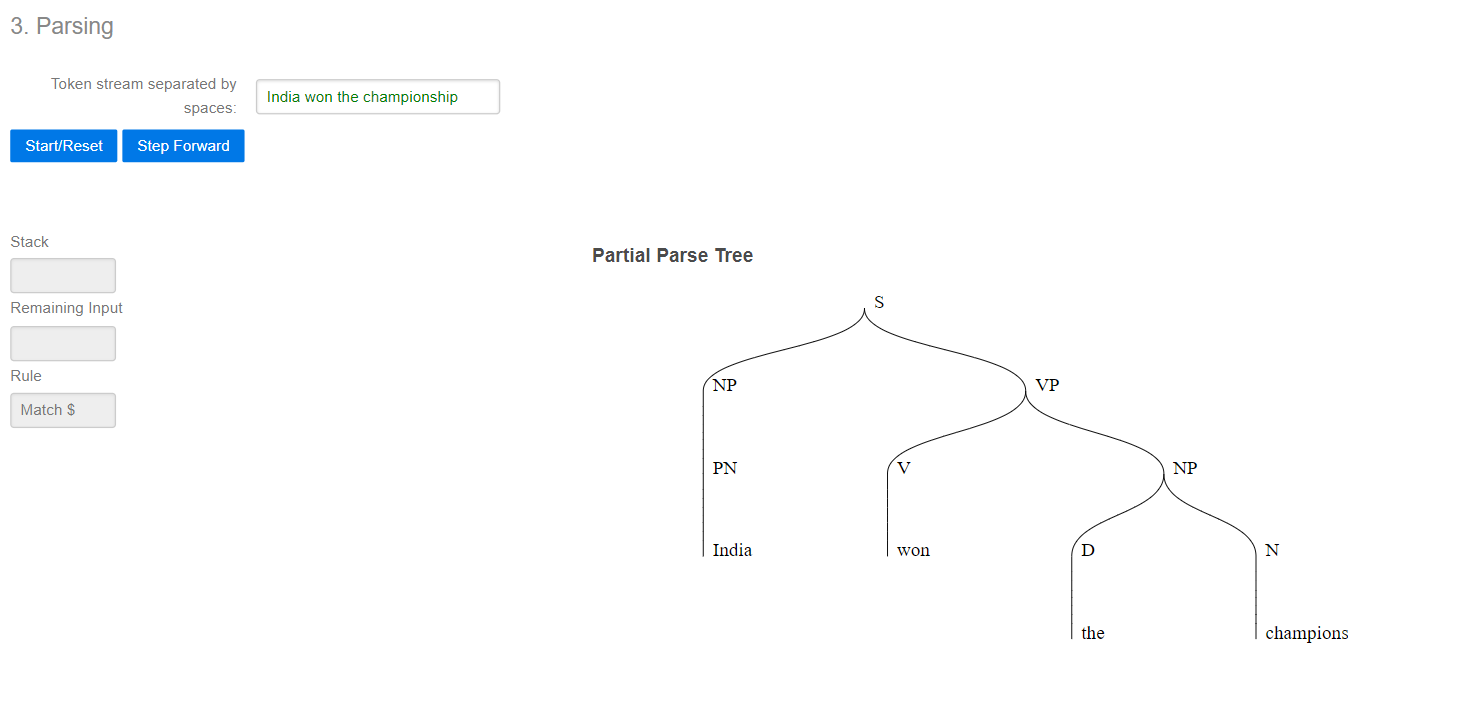
****

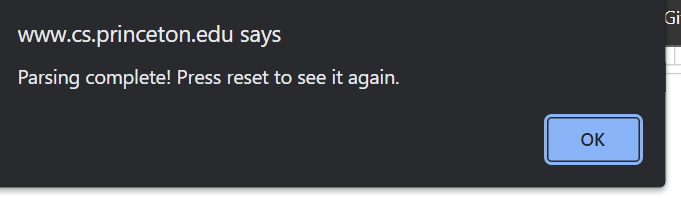
****

****

****

****

****

****