TUGAS BESAR
TEORI BAHASA DAN AUTOMATA
LEXICAL ANALYSIS DAN PARSER SEDERHANA UNTUK TEKS BAHASA ALAMI

Disusun oleh:

Johanes Raphael Nandaputra - 1301204243
Muhammad Naufal Abdillah - 1301201586
Ricardo Hamonangan - 1301204201

Semester Genap 2021/2022

Program Studi S1 Informatika
Fakultas Informatika
Universitas Telkom
Bandung

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Teori bahasa membicarakan bahasa formal (*formal language*), terutama untuk kepentingan perancangan kompilator (*compiler*) dan pemroses naskah (*text processor*). Bahasa formal adalah kumpulan *kalimat*. Semua kalimat dalam sebuah bahasa dibangkitkan oleh sebuah tata bahasa (*grammar*) yang sama.

Automata adalah mesin abstrak yang dapat mengenali (*recognize*), menerima (*accept*), atau membangkitkan (*generate*) sebuah kalimat dalam bahasa tertentu.

Teori Bahasa dan Automata adalah bahasa sebagai input oleh suatu mesin otomata, selanjutnya mesin otomata akan membuat keputusan yang mengindikasikan apakah input itu diterima atau tidak.

1.2 Rumusan Masalah

Berdasarkan latar belakang dan deskripsi tugas besar teori bahasa dan automata maka dapat dirumuskan masalah yang ada pada tugas ini adalah bagaimana cara mengimplementasikan *Context Free Grammar, Finite Automata* kedalam *Lexical Analyzer* dan *Parser*.

1.3 Tujuan

Tujuan ingin dicapai adalah untuk dapat mengimplementasikan *Context Free Grammar, Finite Automata* ke dalam *Lexical Analyzer* dan *Parser*.

BAB II
KAJIAN PUSTAKA

2.1. Context Free Grammar

Context Free Grammar atau dapat disingkat menjadi CFG adalah suatu notasi formal untuk menyatakan definisi rekursif dari suatu bahasa yang menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa. CFG dapat disebut juga sebagai Tata Bahasa Bebas Konteks, dimana tidak terdapat pembatasan pada hasil produksinya dan dapat berbentuk sangat melebar, sangat menyempit, atau terjadi rekursif kiri, yang semuanya sering dinamakan bentuk tidak formal.

2.2. Lexical Analysis

Lexical Analysis adalah sebuah proses yang mendahului parsing sebuah rangkaian karakter yang dilakukan di tahapan pertama pada compiler, ia menerima masukan serangkaian karakter dan menghasilkan deretan simbol yang masing-masing dinamakan token, proses parsing akan lebih mudah dilakukan bila inputnya sudah berupa token. *Lexical Analysis* juga disebut sebagai scanner, yang berarti dapat mengubah deretan karakter-karakter menjadi deretan token-token. Proses pada tahap ini adalah membaca program sumber karakter per karakter. Satu atau lebih deretan karakter ini dikelompokkan menjadi suatu kesatuan mengikuti pola kesatuan kelompok karakter (token) yang ditentukan dalam bahasa sumber dan disimpan dalam tabel simbol, sedangkan karakter yang tidak mengikuti pola akan dilaporkan sebagai token yang tidak valid.

2.3. Parser

Parser adalah komponen *compiler* atau juru bahasa memecah data menjadi elemen yang lebih kecil untuk memudahkan terjemahan ke bahasa lain. *Parsing* adalah suatu cara memecah-mecah suatu rangkaian masukan dengan mengambil input dalam bentuk urutan token atau intruksi program dan menghasilkan struktur data dalam bentuk pohon uraian (*parse*) atau pohon sintaksis abstrak yang akan digunakan pada tahap kompilasi berikutnya yaitu analisis semantik.

BAB III
ANALISIS DAN PERANCANGAN

3.1. Context Free Grammar

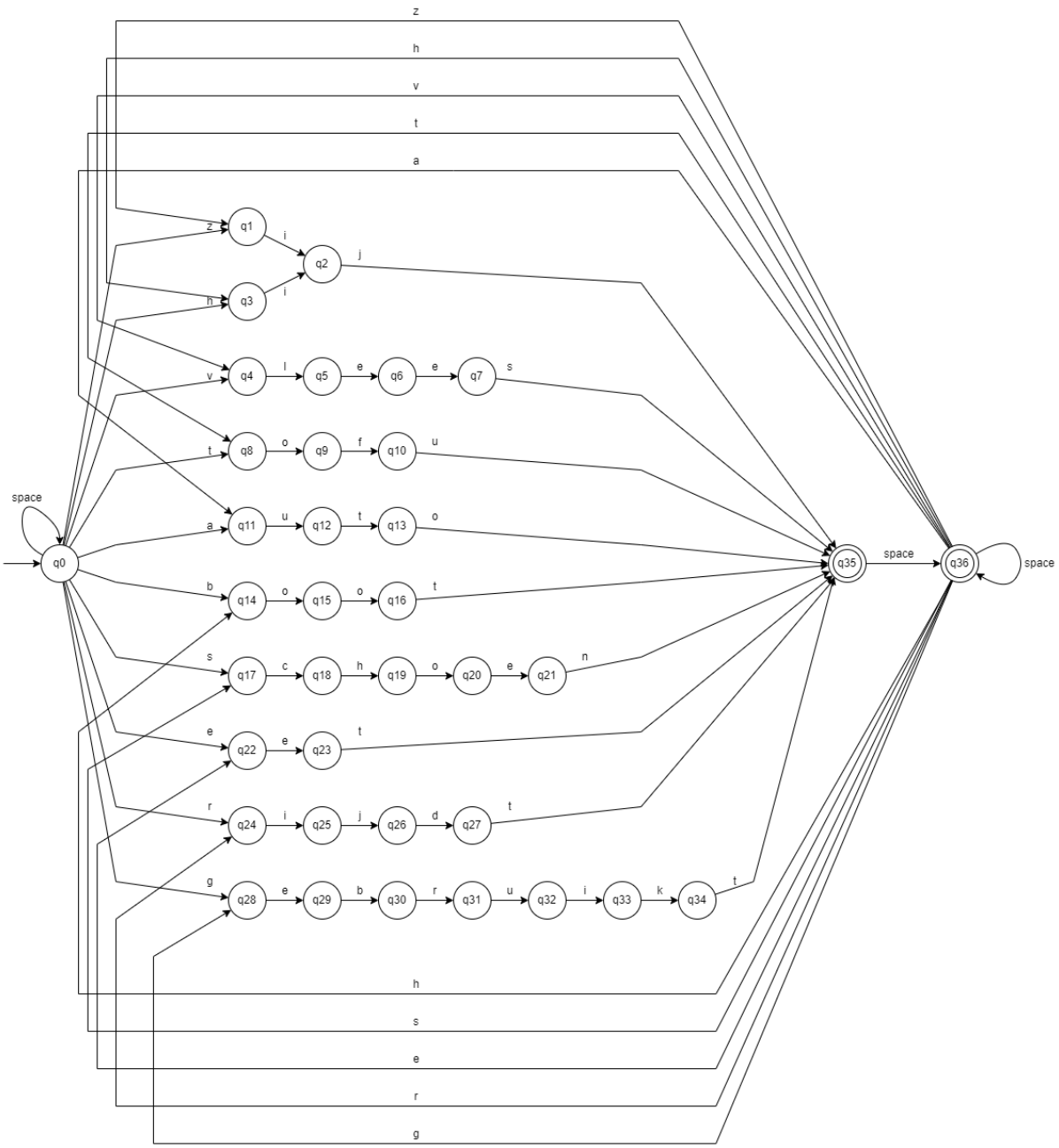
Deskripsi CFG untuk Bahasa Belanda

S -> NN VB NN

NN -> zij | hij | vlees | tofu | auto | boot | schoen

VB -> eet | rijdt | gebruikt

3.2. Finite Automata



3.3. Parse Table LL (1)

	zij	hij	eet	rijdt	gebruikt	vlees	tofu	auto	boot	schoen	EOS
S	NN VB NN	NN VB NN	error	error	error	NN VB NN	NN VB NN	NN VB NN	NN VB NN	NN VB NN	error
NN	zij	hij	error	error	error	vlees	tofu	auto	boot	schoen	error
VB	error	error	eet	rijdt	gebruikt	error	error	error	error	error	error

BAB IV
HASIL

4.1. Lexical Analysis

Berikut merupakan source code dari Lexical Analysis nya:

```
def LA(sentence):
  # init
  global state
  alphabet_list = list(string.ascii_lowercase)
  state_list = [
                'q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9',
'q10',
                'q11', 'q12', 'q13', 'q14', 'q15', 'q16', 'q17', 'q18', 'q19',
'q20',
                'q21', 'q22', 'q23', 'q24', 'q25', 'q26', 'q27', 'q28', 'q29',
'q30',
                'q31', 'q32', 'q33', 'q34', 'q35', 'q36',
              ]
  transition_table = {}

  number_list = ['1','2','3','4','5','6','7','8','9','0']
  for state in state_list:
    for alphabet in alphabet_list:
      transition_table[(state, alphabet)] = 'error'
    for number in number_list:
      transition_table[(state, number)] = 'error'
    transition_table[(state, '#')] = 'error'
    transition_table[(state, ' ')] = 'error'

  #state awal
  transition_table[("q0", " ")] = "q0"

  #state finish
  transition_table[("q35", "#")] = "accept"
  transition_table[("q35", " ")] = "q36"

  transition_table[('q36', '#')] = "accept"
  transition_table[('q36', ' ')] = 'q36'

  #string 'zij'
  transition_table[('q36', 'z')] = 'q1'
  transition_table[('q0', 'z')] = 'q1'
  transition_table[('q1', 'i')] = 'q2'
  transition_table[('q2', 'j')] = 'q35'
  transition_table[('q35', ' ')] = 'q36'

  #string 'hij'
  transition_table[('q36', 'h')] = 'q3'
```

```
transition_table[('q0', 'h')] = 'q3'
transition_table[('q3', 'i')] = 'q2'
transition_table[('q2', 'j')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'vlees'
transition_table[('q36', 'v')] = 'q4'
transition_table[('q0', 'v')] = 'q4'
transition_table[('q4', 'l')] = 'q5'
transition_table[('q5', 'e')] = 'q6'
transition_table[('q6', 'e')] = 'q7'
transition_table[('q7', 's')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'tofu'
transition_table[('q36', 't')] = 'q8'
transition_table[('q0', 't')] = 'q8'
transition_table[('q8', 'o')] = 'q9'
transition_table[('q9', 'f')] = 'q10'
transition_table[('q10', 'u')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'auto'
transition_table[('q36', 'a')] = 'q11'
transition_table[('q0', 'a')] = 'q11'
transition_table[('q11', 'u')] = 'q12'
transition_table[('q12', 't')] = 'q13'
transition_table[('q13', 'o')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'boot'
transition_table[('q36', 'b')] = 'q14'
transition_table[('q0', 'b')] = 'q14'
transition_table[('q14', 'o')] = 'q15'
transition_table[('q15', 'o')] = 'q16'
transition_table[('q16', 't')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'schoen'
transition_table[('q36', 's')] = 'q17'
transition_table[('q0', 's')] = 'q17'
transition_table[('q17', 'c')] = 'q18'
transition_table[('q18', 'h')] = 'q19'
transition_table[('q19', 'o')] = 'q20'
transition_table[('q20', 'e')] = 'q21'
transition_table[('q21', 'n')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'eet'
transition_table[('q36', 'e')] = 'q22'
transition_table[('q0', 'e')] = 'q22'
```

```
transition_table[('q22', 'e')] = 'q23'
transition_table[('q23', 't')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'rijdt'
transition_table[('q36', 'r')] = 'q24'
transition_table[('q0', 'r')] = 'q24'
transition_table[('q24', 'i')] = 'q25'
transition_table[('q25', 'j')] = 'q26'
transition_table[('q26', 'd')] = 'q27'
transition_table[('q27', 't')] = 'q35'
transition_table[('q35', ' ')] = 'q36'

#string 'gebruikt'
transition_table[('q36', 'g')] = 'q28'
transition_table[('q0', 'g')] = 'q28'
transition_table[('q28', 'e')] = 'q29'
transition_table[('q29', 'b')] = 'q30'
transition_table[('q30', 'r')] = 'q31'
transition_table[('q31', 'u')] = 'q32'
transition_table[('q32', 'i')] = 'q33'
transition_table[('q33', 'k')] = 'q34'
transition_table[('q34', 't')] = 'q35'
transition_table[('q35', ' ')] = 'q36'


#Lexical Analysis
idx = 0
state = 'q0'
current = ''
input_string = sentence.lower() + "#"
while state != 'accept':
  current_char = input_string[idx]
  current += current_char
  state = transition_table[(state, current_char)]
  if state == 'q35':
    print("kata saat ini: ", current, ', valid')
  if state == 'error':
    print('error in: ', current, "\n id: ", idx)
    break;
  idx = idx + 1

if state == 'accept':
  print('semua kata di input: ', sentence, ', valid')

return LA
```

Hasil Pengujian Lexical Analysis:

Masukkan Kalimat Belanda:

zij eet vlees

Cek Kata

zij --> valid

zij eet --> valid

zij eet vlees --> valid

zij eet vlees --> all valid

Masukkan Kalimat Belanda:

saya

Cek Kata

error in: sa with the id of `1` and state of q17

schoen (shoe)

Masukkan Kalimat Belanda:

makan daging

Cek Kata

error in: m with the id of `0` and state of q0

schoen (shoe)

Masukkan Kalimat Belanda:

sup

Cek Kata

error in: su with the id of `1` and state of q17

PARSER

4.2. Parser

Kami menggabungkan program *Lexical Analyzer* dengan program *Parser* menjadi satu program. Program ini meminta input berupa kalimat sebanyak tiga kata sesuai *grammar* yang telah ditetapkan dengan struktur NN-VB-NN dalam bahasa Belanda. Berikut merupakan source code dari Parser nya:

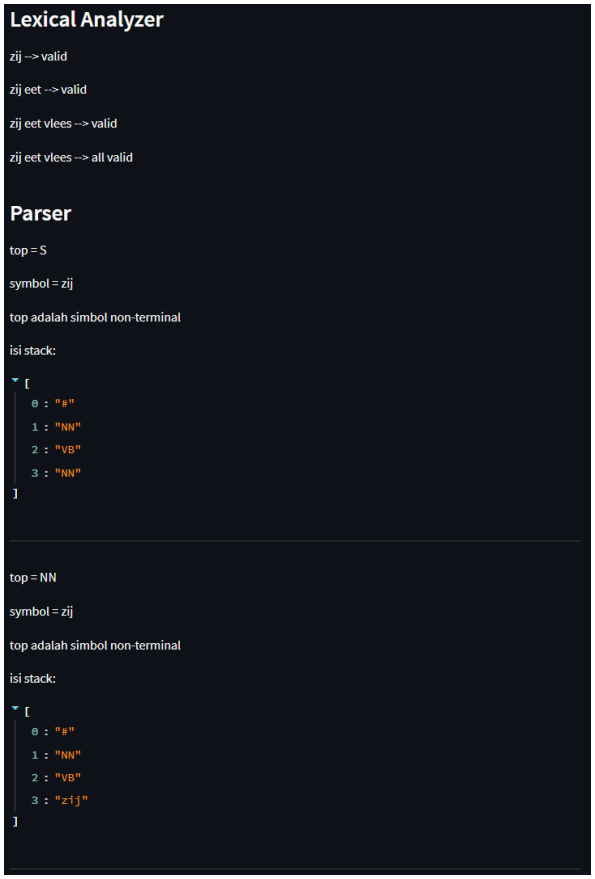
```
def Parser(sentence):
  print("PARSER")
  tokens = sentence.lower().split()
  tokens.append('EOS')

  # definisi simbol
  non_terminals = ['S', 'NN', 'VB']
  terminals = ['zij', 'hij', 'vlees', 'tofu', 'auto', 'boot', 'schoen', 'eet',
'rijdt', 'gebruikt']

  parse_table = {}

  parse_table[('S', 'zij')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'hij')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'vlees')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'tofu')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'auto')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'boot')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'schoen')] = ['NN', 'VB', 'NN']
  parse_table[('S', 'eet')] = ['error']
  parse_table[('S', 'rijdt')] = ['error']
  parse_table[('S', 'gebruikt')] = ['error']
  parse_table[('S', 'EOS')] = ['error']

  parse_table[('NN', 'zij')] = ['zij']
  parse_table[('NN', 'hij')] = ['hij']
  parse_table[('NN', 'vlees')] = ['vlees']
  parse_table[('NN', 'tofu')] = ['tofu']
  parse_table[('NN', 'auto')] = ['auto']
  parse_table[('NN', 'boot')] = ['boot']
  parse_table[('NN', 'schoen')] = ['schoen']
  parse_table[('NN', 'eet')] = ['error']
  parse_table[('NN', 'rijdt')] = ['error']
  parse_table[('NN', 'gebruikt')] = ['error']
  parse_table[('NN', 'EOS')] = ['error']

  parse_table[('VB', 'zij')] = ['error']
  parse_table[('VB', 'hij')] = ['error']
  parse_table[('VB', 'vlees')] = ['error']
  parse_table[('VB', 'tofu')] = ['error']
  parse_table[('VB', 'auto')] = ['error']
  parse_table[('VB', 'boot')] = ['error']
  parse_table[('VB', 'schoen')] = ['error']
  parse_table[('VB', 'eet')] = ['eet']
```

```
parse_table[('VB', 'rijdt')] = ['rijdt']
parse_table[('VB', 'gebruikt')] = ['gebruikt']
parse_table[('VB', 'EOS')] = ['error']

# inisialisasi stack
stack = []
stack.append('#')
stack.append('S')

# inisialisasi input reading
idx_token = 0
symbol = tokens[idx_token]

# proses table parse
while (len(stack) > 0):
  top = stack[len(stack)-1]
  print('top = ', top)
  print('symbol  = ', symbol)
  if top in terminals:
    print('top adalah simbol terminal')
    if top == symbol:
      stack.pop()
      idx_token = idx_token + 1
      symbol = tokens[idx_token]
      if symbol == "EOS":
        stack.pop()
        print('isi stack:', stack)
    else:
      print('error')
      break
  elif top in non_terminals:
    print('top adalah simbol non-terminal')
    if parse_table[(top, symbol)][0] != 'error':
      stack.pop()
      symbol_to_be_pushed = parse_table[(top, symbol)]
      for i in range(len(symbol_to_be_pushed)-1,-1,-1):
        stack.append(symbol_to_be_pushed[i])
    else:
      print('error')
      break
  else:
    print('error')
    break
  print('isi stack: ', stack)
  print()

# kesimpulan
print()
if symbol == 'EOS' and len(stack) == 0:
  print('Input string ', '"', sentence, '"', ' diterima, sesuai Grammar')
else:
```

```
    print('Error, input string:', '"', sentence, '"', ', tidak diterima, tidak
sesuai Grammar')
  return Parser
```

Hasil Pengujian Parser:

1. zij eet vlees (dia makan daging)

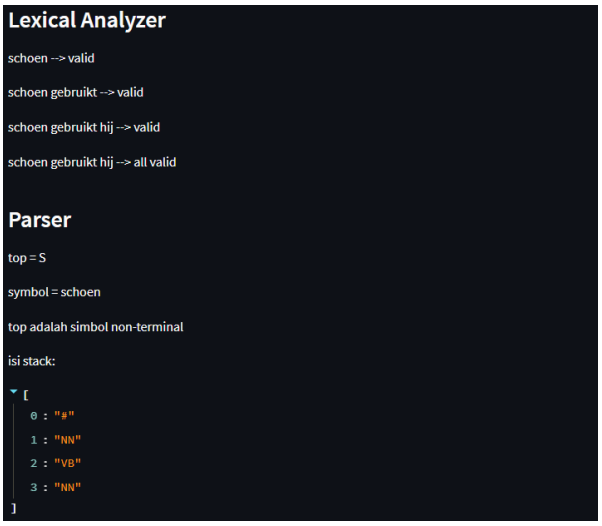


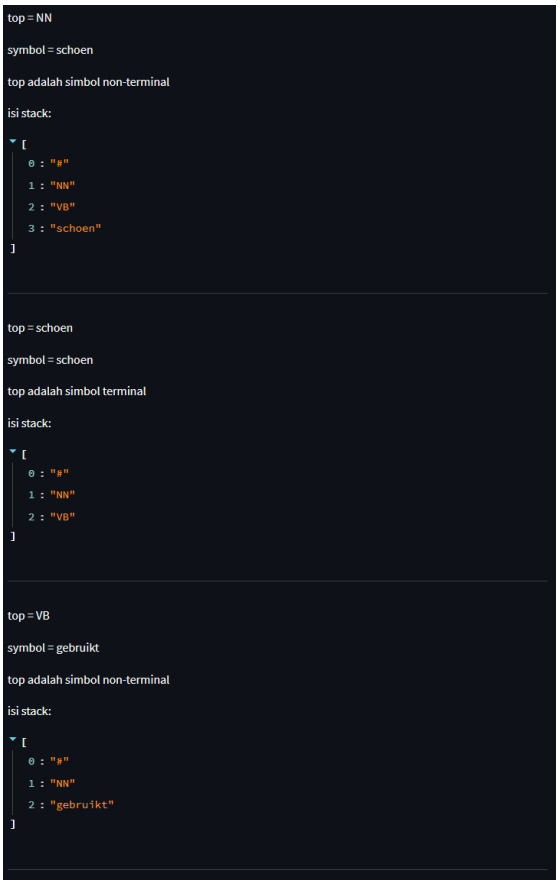
```
top = zij

symbol = zij

top adalah simbol terminal

isi stack:

▾ [
    0 : "#"
    1 : "NN"
    2 : "VB"
]

---

top = VB

symbol = eet

top adalah simbol non-terminal

isi stack:

▾ [
    0 : "#"
    1 : "NN"
    2 : "eet"
]

---

top = eet

symbol = eet

top adalah simbol terminal

isi stack:

▾ [
    0 : "#"
    1 : "NN"
]

---
```

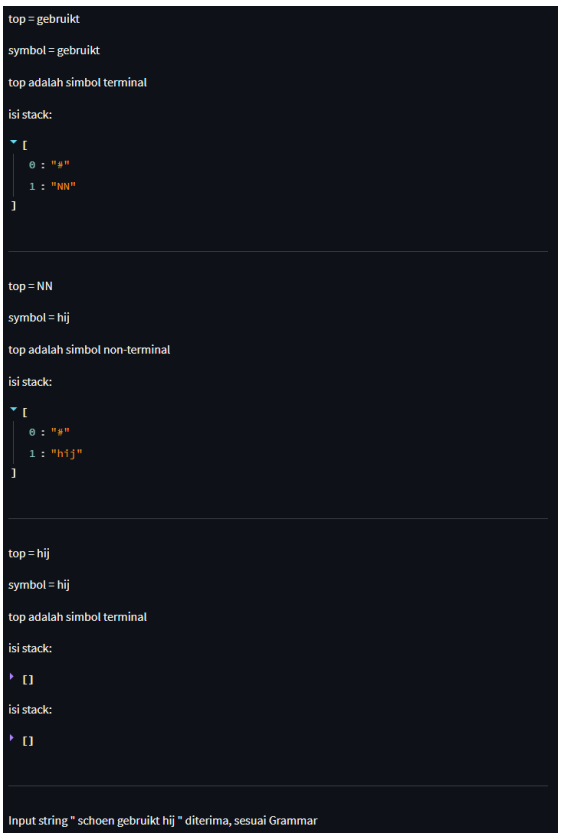
```
top = NN

symbol = vlees

top adalah simbol non-terminal

isi stack:

▾ [
    0 : "#"
    1 : "vlees"
]

---

top = vlees

symbol = vlees

top adalah simbol terminal

isi stack:

▸ []

isi stack:

▸ []

---

Input string " zij eet vlees " diterima, sesuai Grammar
```

Hasil tersebut valid, karena strukturnya berupa NN-VB-NN

2. schoen gebruikt hij (sepatu digunakan dia)

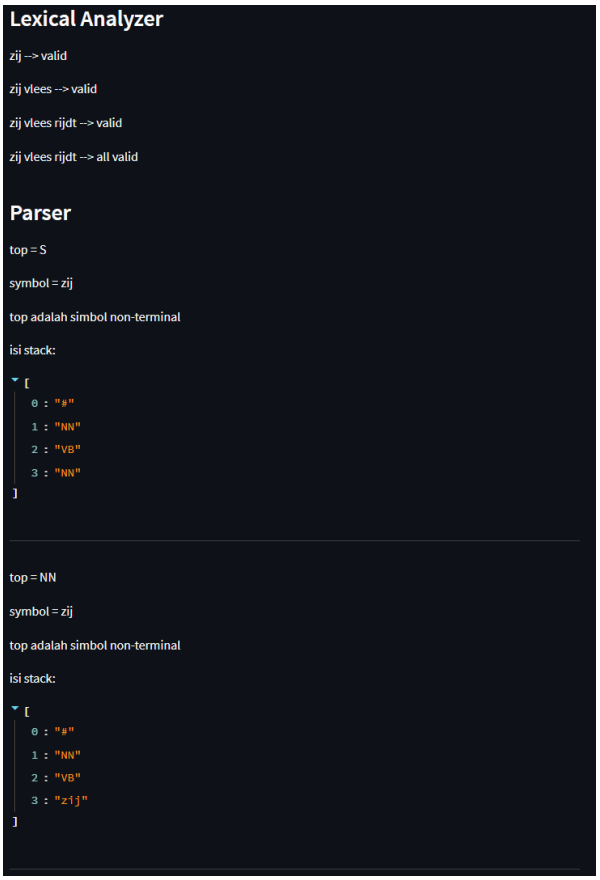


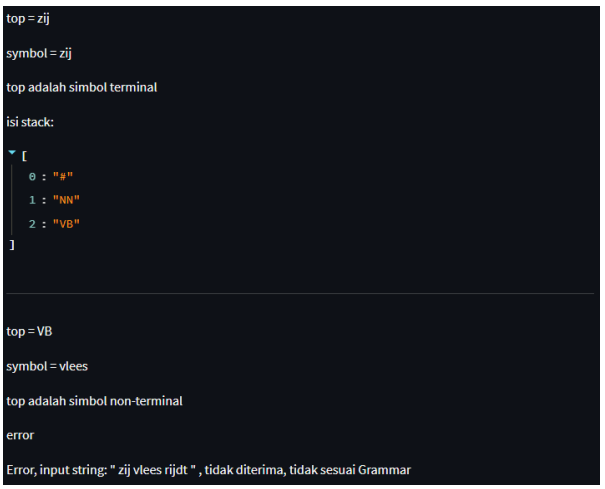




Hasil tersebut valid, karena strukturnya berupa NN-VB-NN walaupun non-terminal dari terminal NN dibalik urutannya.

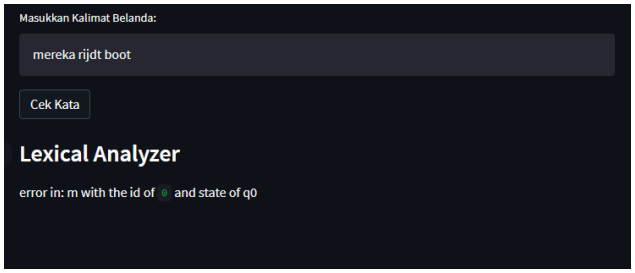
3. zij vlees rijdt (dia daging mengendarai)





Tidak Valid, karena strukturnya berupa NN-NN-VB

4. mereka rijdt boot (mereka kendarai perahu)



Program langsung berhenti pada Lexical Analyzer karena terdapat kata yang tidak terdefinisi, yaitu ‘mereka’

Lampiran

Link website:
https://share.streamlit.io/naufalabdillah1908/tubestba/main/main.py

Link Collab:
https://colab.research.google.com/drive/1Ytz1IuqedU54UixlsB-o-0Hab0sRZma5?usp=sharing#scrollTo=GyJUcSgGZ7XJ

Link Github:
https://github.com/naufalabdillah1908/tubesTBA

Daftar Pustaka

Collins, Michael. “Context Free Grammars”, http://aritter.github.io/courses/5525_slides/cfg.pdf

Aulia, Alvina. 2019. “Teknik Kompilasi: Tahapan Kompilasi”, https://socs.binus.ac.id/2019/12/23/teknik-kompilasi-first-set-pada-top-down-parsing/

Aulia, Alvina. 2018. “Penyederhanaan Context Free Grammar”, https://socs.binus.ac.id/2018/12/20/penyederhanaan-context-free-grammar/