Laporan Tugas Besar

Compiler Bahasa Python

Mata Kuliah IF2124 - Teori Bahasa Formal dan Otomata

Kelompok 5 Kelas 2

Nama Anggota:

Rahmat Rafid Akbar 13502090

Firizky Ardiansyah 13520095

Roby Purnomo 13520106

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021/2022

Daftar Isi

Bab I : Teori Dasar

Finite Automata

Finite Automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata di mana sistem dapat berada di salah satu dari sejumlah berhingga konfigurasi internal disebut state. Beberapa contoh sistem dengan state berhingga antara lain pada mesin minuman otomatis atau *vending machine*, pengatur lampu lalu lintas dan *lexical analyser*.

Suatu finite automata terdiri dari beberapa bagian. Finite automata mempunyai sekumpulan state dan aturan-aturan untuk berpindah dari state yang satu ke state yang lain, tergantung dari simbol nya. Finite automata mempunyai state awal, sekumpulan state dan state akhir. Finite automata merupakan kumpulan dari lima elemen atau dalam bahasa matematis dapat disebut sebagai 5-tuple. Definisi formal dari finite automata dikatakan bahwa finite automata merupakan list dari 5 komponen : kumpulan state, input , aturan perpindahan, state awal, dan state akhir.

Dalam DFA sering digunakan istilah fungsi transisi untuk mendefinisikan aturan perpindahan, biasanya dinotasikan dengan δ. Jika finite automata memiliki sebuah panah dari suatu state x ke suatu state y,dan memiliki label dengan simbol input 0, ini berarti bahwa, jika automata berada pada state x ketika automata tersebut membaca 0, maka automata tersebut dapat berpindah ke state y dapat diindikasikan hal yang sama dengan fungsi transisi dengan mengatakan bahwa δ(x, 0) = y.

Sebuah finite automata terdiri dari lima komponen (Q, Σ, δ, q0, F), di mana :

a. Q adalah himpunan set berhingga yang disebut dengan himpunan states.
b. $\sum$ adalah himpunan berhingga alfabet dari simbol .
c. $\delta : Q \times \sum$ adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state. Fungsi transisi sering dilambangkan dengan δ.
d. $q_0 \subseteq Q$ adalah states awal.
e. $F \subseteq Q$ adalah himpunan states akhir.

Context-Free Grammar

Dalam teori bahasa formal, Context-Free Grammar(CFG) adalah sebuah tata bahasa formal dengan bentuk

$$A \rightarrow \alpha$$

Dengan A adalah sebuah symbol non-terminal, dan α adalah terminal dan atau nonterminal. Context-Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti tata bahasa regular yaitu menunjukkan bagaimana menghasilkan suatu bagian-bagian (untai) dalam sebuah bahasa. Context-Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree* / *parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan. Contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

Dengan penurunan sebagai berikut, akan menghasilkan aabbaa :

$$S \rightarrow aSa \rightarrow aaSaa \rightarrow aabSbaa \rightarrow aabbaa$$

Chomsky Normal Form

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk Context-Free Grammar (CFG) . Bentuk normal Chomsky dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit, dan ε. Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi bentuk normal Chomsky dengan syarat tata bahasa bebas kontesk tersebut:

a. Tidak memiliki produksi useless
b. Tidak memiliki produksi unit
c. Tidak memiliki produksi ε

Bentuk normal Chomsky (Chomsky Normal Form, CNF) adalah Context-Free Grammar (CFG) dengan setiap produksinya berbentuk :

$A \rightarrow BC$ atau $A \rightarrow a$

Bab II : Hasil

Finite Automata

```
1. import string
2. def enumAtoms():
3.     with open('./src/grammar/syntax-atoms.txt', 'r') as f:
4.         lines = f.read()
5.     synatom = lines.split('\n')
6.     enum = dict(zip(synatom, [chr(ord('A')+i) for i in
   range(len(synatom))]))
7.     return enum
8.
9. def read(filepath: str):
10.    f = open(filepath, 'r')
11.    w = f.read()
12.    alp = string.ascii_lowercase
13.    Alp = string.ascii_uppercase
14.    num = string.digits
15.    res = ""
16.    atom = enumAtoms()
17.
18.    while(w):
19.        # string processing
20.        if(w[0:3]=="\'\'\'"):
21.            nl = 0
22.            tmp = w[3:]
23.            while(tmp):
24.                if(tmp[0]=='\n'):
25.                    nl+=1
26.                if(tmp[0:3]=="\'\'\'"):
27.                    break
28.                tmp = tmp[1:]
29.            if(tmp):
30.                if(tmp[0:3]=="\'\'\'"):
31.                    for _ in range(nl):
32.                        res += 's\n'
33.                    res += 's'
34.                    w = tmp[2:]
35.        elif(w[0:3]=="\"\"\""):
36.            nl = 0
37.            tmp = w[3:]
38.            while(tmp):
39.                if(tmp[0]=='\n'):
40.                    nl+=1
```

```
                if(tmp[0:3]=="\"\"\""):
                    break
                tmp = tmp[1:]
            if(tmp):
                if(tmp[0:3]=="\"\"\""):
                    for _ in range(nl):
                        res += 's\n'
                    res += 's'
                    w = tmp[2:]
        elif(w[0]=='\"'):
            tmp = w[1:]
            while(tmp):
                if(tmp[0]=='\n' or tmp[0]=='\"'):
                    break
                tmp = tmp[1:]
            if(tmp):
                if(tmp[0]=='\n'):
                    res += w[0]
                else:
                    res += 's'
                    w = tmp
            else:
                res += w[0]
        elif(w[0]=='\''):
            tmp = w[1:]
            while(tmp):
                if(tmp[0]=='\n' or tmp[0]=='\''):
                    break
                tmp = tmp[1:]
            if(tmp):
                if(tmp[0]=='\n'):
                    res += w[0]
                else:
                    res += 's'
                    w = tmp
            else:
                res += w[0]
        # cek comment
        elif(w[0] == '#'):
            tmp = w
            w = w[1:]
            while(w):
                if(w[0]=='\n'):
                    break
                tmp = w
                w = w[1:]
            if(not w):
                break
```

```
            else:
                w = tmp
        elif(w[0] not in (alp+Alp+num+'_')):
            res += w[0]
        else:
            if(w[0] not in (alp+Alp+'_')):
                # cek floating point dan integer kemudian disubstitusi menjadi 'n'
                # cek ada var tidak valid, kemudian di assign menjadi 'x'
                if(w[0] in num):
                    tmp = w
                    w = w[1:]
                        while(w):
                            if(w[0] not in num):
                                break
                            tmp = w
                            w = w[1:]
                        if(not w):
                            res += 'n'
                            break
                        if(w[0]=='.'):
                            tmp = w
                            w = w[1:]
                            while(w):
                                if(w[0] not in num):
                                    break
                                tmp = w
                                w = w[1:]
                            if(not w):
                                res += 'n'
                                break
                            if(w[0] in (alp+Alp+'_')):
                                tmp = w
                                w = w[1:]
                                while(w):
                                    if(w[0] not in (alp+Alp+'_'+num)):
                                        break
                                    tmp = w
                                    w = w[1:]
                                if(not w):
                                    res += 'n.x'
                                    break
                                w = tmp
                                res += 'n.x'
                            else:
                                res += 'n'
```

```
                                    w = tmp
                                elif(w[0] in (alp+Alp+'_')):
                                    tmp = w
                                    w = w[1:]
                                    while(w):
                                        if(w[0] not in (alp+Alp+'_'+num)):
                                            break
                                        tmp = w
                                        w = w[1:]
                                    if(not w):
                                        res += 'x'
                                        break
                                    w = tmp
                                    res += 'x'
                                else:
                                    w = tmp
                                    res += 'n'
                            elif(w[0]=='.'):
                                tmp = w
                                w = w[1:]
                                while(w):
                                    if(w[0] not in num):
                                        break
                                    tmp = w
                                    w = w[1:]
                                if(not w):
                                    res += 'n'
                                    break
                                if(w[0] in (alp+Alp+'_')):
                                    tmp = w
                                    w = w[1:]
                                    while(w):
                                        if(w[0] not in (alp+Alp+'_'+num)):
                                            break
                                        tmp = w
                                        w = w[1:]
                                    if(not w):
                                        res += 'x'
                                        break
                                    w = tmp
                                    res += 'x'
                                else:
                                    res += 'n'
                        else:
                            cur = w[0]
                            tmp = w
                            w = w[1:]
```

```
                    while w:
                        if(w[0] not in (alp+Alp+num+'_')):
                            break
                        else:
                            cur += w[0]
                        tmp = w
                        w = w[1:]
                    if(cur in atom):
                        if(w):
                            res += atom[cur]
                            w = tmp
                        else:
                            res += atom[cur]
                            w = tmp
                    else:
                        if(w):
                            res += 'v'
                            w = tmp
                        else:
                            res+='v'
                            w = tmp
            tmp = w
            w = w[1:]
        return res.replace(" ", "")
```

Context-Free Grammar

```
START -> STATEMENTS | NEWLINE | NEWLINE START | START NEWLINE
STATEMENTS -> STATEMENT NEWLINE STATEMENTS | NEWLINE STATEMENTS | STATEMENTS
NEWLINE | STATEMENT
STATEMENT -> SIMPLE_STATEMENT | COMPOUND_STATEMENT
COMPOUND_STATEMENT -> DEF_STATEMENT | IF_STATEMENT | CLASS_STATEMENT |
WITH_STATEMENT | FOR_STATEMENT | WHILE_STATEMENT
SIMPLE_STATEMENT -> ASSIGNMENT | STAR_EXPRESSIONS | RETURN_STATEMENT |
IMPORT_STATEMENT | RAISE_STATEMENT | PASS | BREAK | CONTINUE
BLOCK -> NEWLINE STATEMENTS | SIMPLE_STATEMENT

ATOM -> NAME | TRUE | FALSE | NONE | STRINGS | NUMBER | TUPLE | GROUP |
GENERAL_EXPRESSION | DICT | LIST | SET

// Index processing
SLICE -> EXPRESSION | ASSIGNMENT_EXPRESSION | COLON | EXPRESSION COLON | COLON
EXPRESSION | EXPRESSION COLON EXPRESSION | COLON COLON | EXPRESSION COLON
```

```
COLON | COLON EXPRESSION COLON | EXPRESSION COLON EXPRESSION COLON | COLON
COLON EXPRESSION | EXPRESSION COLON COLON EXPRESSION | COLON EXPRESSION COLON
EXPRESSION | EXPRESSION COLON EXPRESSION COLON EXPRESSION
SLICES -> SLICE COMMA SLICES | SLICE

// Argument fungsi
KWARG_STARRED -> NAME EQ EXPRESSION | STAR EXPRESSION
KWARG_DOUBLE_STARRED -> NAME EQ EXPRESSION | STAR STAR EXPRESSION
KWARGS_STARRED -> KWARG_STARRED COMMA KWARGS_STARRED | KWARG_STARRED
KWARGS_DOUBLE_STARRED -> KWARG_DOUBLE_STARRED COMMA KWARGS_DOUBLE_STARRED |
KWARG_DOUBLE_STARRED
KWARGS -> KWARGS_STARRED | KWARGS_DOUBLE_STARRED | KWARG_STARRED COMMA
KWARGS_DOUBLE_STARRED
NORM_ARGUMENT -> STAR EXPRESSION COMMA NORM_ARGUMENT | ASSIGNMENT_EXPRESSION
COMMA NORM_ARGUMENT | EXPRESSION COMMA NORM_ARGUMENT | STAR EXPRESSION |
ASSIGNMENT | EXPRESSION
ARGS -> NORM_ARGUMENT | KWARGS | NORM_ARGUMENT COMMA KWARGS
ARGUMENTS -> ARGS COMMA ARGUMENTS | ARGS

// object
OBJECT -> OBJECT DOT NAME | OBJECT GENERAL_EXPRESSION | OBJECT L_BRACHET
ARGUMENTS R_BRACHET | OBJECT LS_BRACHET SLICES RS_BRACHET | ATOM
STAR_OBJECT -> LIST COMMA STAR_OBJECT | TUPLE COMMA STAR_OBJECT | OBJECT COMMA
STAR_OBJECT | STAR OBJECT COMMA STAR_OBJECT | LIST | TUPLE | OBJECT | STAR
OBJECT
LIST -> LS_BRACHET STAR_NAMED_EXPRESSIONS RS_BRACHET | LS_BRACHET RS_BRACHET
TUPLE -> L_BRACHET STAR_NAMED_EXPRESSIONS R_BRACHET | L_BRACHET R_BRACHET
GROUP -> L_BRACHET NAMED_EXPRESSION R_BRACHET
SET -> LC_BRACHET STAR_EXPRESSIONS RC_BRACHET
PAIR -> STAR STAR LOGIC_OBJECT | EXPRESSION COLON EXPRESSION
PAIRS -> PAIR COMMA PAIRS | PAIR
DICT -> LC_BRACHET PAIRS RC_BRACHET | LC_BRACHET RC_BRACHET
STRINGS -> STRING | STRING STRINGS

// Aritmatika
SINGLETON -> UNARY_OP SINGLETON | OBJECT
IDENTIFIER -> NAME | OBJECT | L_BRACHET IDENTIFIER R_BRACHET | OBJECT DOT NAME
| OBJECT LS_BRACHET SLICES RS_BRACHET
TERM -> SINGLETON BINARY_OP TERM | SINGLETON
```

```
ARITHMETIC_OBJECT -> ARITHMETIC_OBJECT PLUS TERM | ARITHMETIC_OBJECT MINUS
TERM | TERM

// Logic
LOGIC_OBJECT -> ARITHMETIC_OBJECT LOGIC_OP LOGIC_OBJECT | ARITHMETIC_OBJECT

// Comparison
COMPARISON_OBJECT -> NOT COMPARISON_OBJECT | LOGIC_OBJECT COMPARISON_OP
COMPARISON_OBJECT | LOGIC_OBJECT
COMPOUND_OBJECT -> COMPARISON_OBJECT BINARY_COMPARISON_OP COMPOUND_OBJECT |
COMPARISON_OBJECT

// EXPRESSION
EXPRESSION -> COMPOUND_OBJECT IF COMPOUND_OBJECT ELSE EXPRESSION |
COMPOUND_OBJECT
EXPRESSIONS -> EXPRESSION COMMA EXPRESSIONS | EXPRESSION
ASSIGNMENT_EXPRESSION -> NAME SPECIAL_ASSIGNMENT EXPRESSION
NAMED_EXPRESSION -> ASSIGNMENT_EXPRESSION | EXPRESSION
STAR_NAMED_EXPRESSION -> STAR LOGIC_OBJECT | NAMED_EXPRESSION
STAR_EXPRESSION -> STAR LOGIC_OBJECT | EXPRESSION
STAR_NAMED_EXPRESSIONS -> STAR_NAMED_EXPRESSION COMMA STAR_NAMED_EXPRESSIONS |
STAR_NAMED_EXPRESSION
STAR_EXPRESSIONS -> STAR_EXPRESSION COMMA STAR_EXPRESSIONS | STAR_EXPRESSION

// Compound EXPRESSION
IF_EXPRESSION -> IF_EXPRESSION IF_EXPRESSION | IF COMPOUND_OBJECT
FOR_IF_EXPRESSION -> FOR STAR_OBJECT IN COMPOUND_OBJECT | FOR STAR_OBJECT IN
COMPOUND_OBJECT IF_EXPRESSION
FOR_IF_EXPRESSIONS -> FOR_IF_EXPRESSION FOR_IF_EXPRESSIONS | FOR_IF_EXPRESSION
GENERAL_EXPRESSION -> NAMED_EXPRESSION FOR_IF_EXPRESSIONS

// statement
ASSIGNMENT -> NAME COLON EXPRESSION | NAME COLON EXPRESSION EQ
STAR_EXPRESSIONS | IDENTIFIER ASSIGNMENT_OP STAR_EXPRESSIONS

// Import
PACKAGE -> PACKAGE DOT NAME | NAME
PACKAGES -> PACKAGES AS NAME COMMA PACKAGES | PACKAGES DOT NAME COMMA PACKAGES
| NAME COMMA PACKAGES| PACKAGES DOT NAME | NAME | PACKAGES AS NAME
FROM_PACKAGES -> STAR | PACKAGES
```

```
IMPORT_STATEMENT -> IMPORT PACKAGES | FROM ELIPSIS PACKAGE IMPORT
FROM_PACKAGES | FROM ELIPSIS IMPORT FROM_PACKAGES | FROM PACKAGE IMPORT
FROM_PACKAGES

// if statement
IF_STATEMENT -> IF NAMED_EXPRESSION COLON BLOCK | IF NAMED_EXPRESSION COLON
BLOCK ELIF_STATEMENT | IF NAMED_EXPRESSION COLON BLOCK ELSE_STATEMENT
ELIF_STATEMENT -> ELIF NAMED_EXPRESSION COLON BLOCK | ELIF NAMED_EXPRESSION
COLON BLOCK ELSE_STATEMENT | ELIF NAMED_EXPRESSION COLON BLOCK ELIF_STATEMENT
ELSE_STATEMENT -> ELSE COLON BLOCK

// while loop
WHILE_STATEMENT -> WHILE NAMED_EXPRESSION COLON BLOCK | WHILE NAMED_EXPRESSION
COLON BLOCK ELSE_STATEMENT

// for loop
FOR_STATEMENT -> FOR STAR_OBJECT IN STAR_EXPRESSIONS COLON BLOCK | FOR
STAR_OBJECT IN STAR_EXPRESSIONS COLON BLOCK ELSE_STATEMENT

// with statement
WITH_STATEMENT -> WITH L_BRACHET WITH_CONTENT R_BRACHET COLON BLOCK | WITH
WITH_CONTENT COLON BLOCK
WITH_CONTENT -> WITH_CONTENT COMMA WITH_CONTENT | EXPRESSION AS STAR_OBJECT |
EXPRESSION

// class statement
CLASS_STATEMENT -> CLASS NAME COLON BLOCK | CLASS NAME L_BRACHET R_BRACHET
COLON BLOCK | CLASS NAME L_BRACHET ARGUMENTS R_BRACHET COLON BLOCK

// raise
RAISE_STATEMENT -> RAISE | RAISE EXPRESSION | RAISE EXPRESSION FROM EXPRESSION

// return
RETURN_STATEMENT -> RETURN | RETURN STAR_EXPRESSIONS

// def
DEF_STATEMENT -> DEF NAME L_BRACHET R_BRACHET COLON BLOCK | DEF NAME L_BRACHET
PARAMS R_BRACHET COLON BLOCK
```

```
// DEF_PARAMS
PARAMS -> PAR_ONE | PAR_ONE STAR_ETC | PAR_TWO | PAR_TWO STAR_ETC | PAR_THREE
| PAR_THREE STAR_ETC | PAR_FOUR | PAR_FOUR STAR_ETC | STAR_ETC
PAR_ONE -> PAR_ONE_SEC | PAR_ONE_SEC PARAM_WITH_DEFAULT | PAR_ONE
PARAM_WITH_DEFAULT
PAR_ONE_SEC -> SLASH_NO_DEFAULT | SLASH_NO_DEFAULT PARAM_NO_DEFAULT |
PAR_ONE_SEC PARAM_NO_DEFAULT
PAR_TWO -> SLASH_WITH_DEFAULT | SLASH_WITH_DEFAULT PARAM_WITH_DEFAULT |
PAR_TWO PARAM_WITH_DEFAULT
PAR_THREE -> PAR_THREE_SEC | PAR_THREE_SEC PARAM_WITH_DEFAULT | PAR_THREE
PARAM_WITH_DEFAULT
PAR_THREE_SEC -> PARAM_NO_DEFAULT PAR_THREE_SEC | PARAM_NO_DEFAULT
PAR_FOUR -> PARAM_WITH_DEFAULT PAR_FOUR | PARAM_WITH_DEFAULT


// SLASH_PARAMS
SLASH_NO_DEFAULT -> SLASH_NO_DEFAULT_ONE SLASH COMMA | SLASH_NO_DEFAULT_ONE
SLASH
SLASH_NO_DEFAULT_ONE -> PARAM_NO_DEFAULT | PARAM_NO_DEFAULT
SLASH_NO_DEFAULT_ONE

SLASH_WITH_DEFAULT -> SLASH_WITH_DEFAULT_ONE | SLASH_WITH_DEFAULT_TWO
SLASH_WITH_DEFAULT_ONE -> SLASH_WITH_DEFAULT_ONE_SEC SLASH COMMA |
PARAM_NO_DEFAULT SLASH_WITH_DEFAULT_ONE_SEC SLASH COMMA | PARAM_NO_DEFAULT
SLASH_WITH_DEFAULT_ONE
SLASH_WITH_DEFAULT_ONE_SEC -> PARAM_WITH_DEFAULT | PARAM_WITH_DEFAULT
SLASH_WITH_DEFAULT_ONE_SEC
SLASH_WITH_DEFAULT_TWO -> SLASH_WITH_DEFAULT_ONE_SEC SLASH | PARAM_NO_DEFAULT
SLASH_WITH_DEFAULT_ONE_SEC SLASH | PARAM_NO_DEFAULT SLASH_WITH_DEFAULT_TWO

STAR_ETC -> STAR_ETC_ONE | STAR_ETC_ONE KWDS | STAR_ETC_TWO | STAR_ETC_TWO
KWDS | KWDS
STAR_ETC_ONE -> STAR PARAM_NO_DEFAULT | STAR PARAM_NO_DEFAULT
PARAM_MAYBE_DEFAULT | STAR_ETC_ONE PARAM_MAYBE_DEFAULT
STAR_ETC_TWO -> STAR COMMA STAR_ETC_TWO_SEC
STAR_ETC_TWO_SEC -> PARAM_MAYBE_DEFAULT | PARAM_MAYBE_DEFAULT STAR_ETC_TWO_SEC
KWDS -> STAR STAR PARAM_NO_DEFAULT

PARAM_NO_DEFAULT -> PARAM COMMA | PARAM
PARAM_WITH_DEFAULT -> PARAM DEFAULT COMMA | PARAM DEFAULT
```

```
PARAM_MAYBE_DEFAULT -> PARAM COMMA | PARAM DEFAULT COMMA | PARAM | PARAM
DEFAULT
PARAM -> NAME | NAME ANNOTATION

ANNOTATION -> COLON EXPRESSION
DEFAULT -> EQ EXPRESSION

ASSIGNMENT_OP -> + = | - = | * = | @ = | / = | % = | & = | OR_OP = | ^ = | < <
= | > > = | * * = | / / = | =
SPECIAL_ASSIGNMENT -> : =
UNARY_OP -> PLUS | MINUS | TILDE
BINARY_OP -> * | / | / / | % | @ | * *
LOGIC_OP -> > > | < < | & | ^ | OR_OP
EXISTENTIAL_OP -> NOT IN | IN
SIMILARITY_OP -> IS NOT | IS
COMPARISON_OP -> < | > = | < | < = | = = | ! = | EXISTENTIAL_OP |
SIMILARITY_OP
BINARY_COMPARISON_OP -> AND | OR
ELIPSIS -> DOT | DOT DOT DOT | ELIPSIS ELIPSIS

COMMA -> ,
COLON -> :
PLUS -> +
MINUS -
TILDE -> ~
STAR -> *
ARROW -> ->
DOT -> .
EQ -> =
SLASH -> /
AT_SIGN -> @
LC_BRACHET -> {
RC_BRACHET -> }
LS_BRACHET -> [
RS_BRACHET -> ]
L_BRACHET -> (
R_BRACHET -> )
NEWLINE -> NEWLINE NEWLINE | __nl__
OR_OP -> __or__
```

```
FALSE -> A
CLASS -> B
IS -> C
RETURN -> D
NONE -> E
CONTINUE -> F
FOR -> G
TRUE -> H
DEF -> I
FROM -> J
WHILE -> K
AND -> L
NOT -> M
WITH -> N
AS -> O
ELIF -> P
IF -> Q
OR -> R
ELSE -> S
IMPORT -> T
PASS -> U
BREAK -> V
IN -> W
RAISE -> X
NAME -> v
NUMBER -> n
STRING -> s
```

Bab III : Implementasi Dan Pengujian

Implementasi pada source code

1. readGrammar

   Fungsi read grammar digunakan untuk membaca grammar yang berasal dari file .txt yang telah dibuat (grammar.txt). Fungsi ini mengembalikan CFG dalam bentuk dictionary (pada python).
2. CFGtoCNF

   Fungsi CFGtoCNF memiliki parameter masukan berupa CFG lalu CFG ini akan dimodifikasi sehingga memenuhi bentuk CNF (Chomsky Normal Form) yang juga menggunakan tipe data dictionary.
3. CYK

   Algoritma CYK disini digunakan untuk memeriksa substring dari string input mana yang dikenali oleh CFG yang diberikan setiap line dari file .py yang akan dievaluasi syntaxnya.
4. Main Program

   Pada main program akan dilakukan pemanggilan fungsi-fungsi diatas, dan akan menghandle input dan output pada program dengan pengguna.

Link Repository : https://github.com/firizky29/TubesTBFO

Testing

1. Testing dari Spek
   a. inputAcc.py

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/inputAcc.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  def do_something(x):
    2 |      ''' This is a sample multiline comment
    3 |      '''
    4 |      if x == 0:
    5 |          return 0
    6 |      elif x + 4 == 1:
    7 |          if True:
    8 |              return 3
    9 |          else:
   10 |              return 2
   11 |      elif x == 32:
   12 |          return 4
   13 |      else:
   14 |          return "Doodoo"

>> Evaluating test/inputAcc.py...

>> Hasil Evaluate
   Accepted

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

   b. inputReject.py

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/inputReject.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  def do_something(x):
    2 |      ''' This is a sample multiline comment
    3 |      '''
    4 |      x + 2 = 3
    5 |      if x == 0 + 1
    6 |          return 0
    7 |      elif x + 4 == 1:
    8 |          else:
    9 |              return 2
   10 |      elif x == 32:
   11 |          return 4
   12 |      else:
   13 |          return "Doodoo"

>> Evaluating test/inputReject.py...

>> Hasil Evaluate
   Syntax Error
   4 |      x + 2 = 3

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

2. Conditional

a. Accepted

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/conditional.py"
================================================================
                   PYTHON SYNTAX EVALUATOR
                 Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  if (x == True and y <20) :
    2 |      x = False
    3 |  elif (y == 20 or x == False) :
    4 |      y = 3.4
    5 |      z = None
    6 |  else :
    7 |      z = 'win'

>> Evaluating test/conditional.py...

>> Hasil Evaluate
   Accepted

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

b. Syntax Error

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/conditional.py"
================================================================
                   PYTHON SYNTAX EVALUATOR
                 Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  if (x == True and y <20) :
    2 |      x = False
    3 |  elif (y = 20 or x == False) :
    4 |      y = 3.4
    5 |      z = None
    6 |  else :
    7 |      z = 'win'

>> Evaluating test/conditional.py...

>> Hasil Evaluate
   Syntax Error
   3 | elif (y = 20 or x == False) :

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

3. Class + def

a. Accepted

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/class.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  class Person:
    2 |      def __init__(self, name, age):
    3 |          self.name = name
    4 |          self.age = age
    5 |
    6 |      def tbfo(soal):
    7 |          return susah


>> Evaluating test/class.py...

>> Hasil Evaluate
   Accepted

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

b. Syntax Error

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/class.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  class Person:
    2 |      def __init__(self, name, age):
    3 |          self.name = name
    4 |          self.age = age
    5 |
    6 |      def tbfo(soal):
    7 |          return susah
    8 |
    9 |  def x :


>> Evaluating test/class.py...

>> Hasil Evaluate
   Syntax Error
   9 | def x :
```

4. Loop (for and while)

   a. Accepted

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/loop.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  a = 1
    2 |
    3 |  for i in range(40):
    4 |      if i == 5 :
    5 |          pass
    6 |      else :
    7 |          continue
    8 |
    9 |  while (a<20) :
   10 |      if a == 5:
   11 |          break
   12 |      a += 1

>> Evaluating test/loop.py...

>> Hasil Evaluate
   Accepted

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

   b. Syntax Error

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/loop.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  a = 1
    2 |
    3 |  for i in range(40):
    4 |      if i == 5 :
    5 |          pass
    6 |      else :
    7 |          continue
    8 |
    9 |  while (a<20) :
   10 |      if a == 5:
   11 |          break
   12 |      a += 1
   13 |
   14 |  for i in [1,2,3] :
   15 |      a *= 1
   16 |
   17 |  while (True)
   18 |      break

>> Evaluating test/loop.py...

>> Hasil Evaluate
   Syntax Error
   17 | while (True)

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

5. Import

   a. Accepted

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

D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/import.py"
============================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
============================================================

>> Source Code
    1 |  from matplotlib import pyplot
    2 |  from src.CFGtoCNF import CFGtoCNF
    3 |  import numpy
    4 |  import flask as f

>> Evaluating test/import.py...

>> Hasil Evaluate
   Accepted

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

   b. Syntax Error

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/import.py"
============================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
============================================================

>> Source Code
    1 |  from matplotlib import pyplot
    2 |  from "src/CFGtoCNF" import CFGtoCNF
    3 |  import numpy
    4 |  import flask as f

>> Evaluating test/import.py...

>> Hasil Evaluate
   Syntax Error
   2 | from "src/CFGtoCNF" import CFGtoCNF

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

6. Other

```
D:\ROBY\tbfo\tubes\syntaxevaluate\TubesTBFO>python main.py "test/other.py"
================================================================
                    PYTHON SYNTAX EVALUATOR
                  Made by : Kelompok 5 Kelas 2
================================================================

>> Source Code
    1 |  with open('file_path', 'w') as file:
    2 |      file.write('hello world !')
    3 |
    4 |  if (not A) :
    5 |    raise Exception("Sorry, no numbers below zero")

>> Evaluating test/other.py...

>> Hasil Evaluate
   Accepted

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

BAB IV : Pembagian Tugas

Anggota	NIM	Tugas
Rahmat Rafid Akbar	13502090	- Membuat CFG - Debugging - Laporan
Firizky Ardiansyah	13520095	- Algoritma read file to grammar - Algoritma CYK - Membuat CFG - Debugging
Roby Purnomo	13520106	- Algoritma convert CFG to CNF - Laporan - Main Program - Debugging

Bab V : Kesimpulan dan Saran

Kesimpulan

a. Kami telah berhasil membuat FA dan CFG untuk mengecek string apakah diterima oleh syntax python.
b. Kami berhasil membuat program dengan python yang memanfaatkan FA dan CFG yang kami buat dengan menggunakan bantuan beberapa fungsi yang telah dipaparkan diatas.

Saran

Dikarenakan dalam tugas besar ini diberikan banyak pengecualian syntax, mungkin kedepannya dapat lebih menyempurnakan tugas besar TBFO seperti penggunaan indentasi, r-string, f-string, dll.