

**LAPORAN TUGAS BESAR**

**Compiler Bahasa Python**

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2124 Teori Bahasa Formal  
dan Otomata pada Semester I Tahun Akademik 2021/2022

Disusun oleh:

<b>Marchotridyo</b>	<b>13520119</b>
<b>Januar Budi Ghifari</b>	<b>13520132</b>
<b>Rania Dwi Fadhillah</b>	<b>13520142</b>



**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2021**

## DAFTAR ISI

<b>DAFTAR ISI .....</b>	<b>i</b>
<b>BAB I TEORI DASAR .....</b>	<b>1</b>
<b>BAB II HASIL.....</b>	<b>4</b>
<b>BAB III IMPLEMENTASI DAN PENGUJIAN .....</b>	<b>11</b>
I. Spesifikasi Teknis Program .....	11
II. Uji Kasus .....	13
<b>BAB IV PEMBAGIAN TUGAS .....</b>	<b>20</b>
<b>LAMPIRAN.....</b>	<b>ii</b>

## BAB I

### TEORI DASAR

#### 1. *Finite Automata*

*Finite automata* merupakan mesin abstrak tersimpel yang dapat digunakan untuk memahami suatu pola tertentu dengan cara mengenali pola dalam *input* yang diambil dari beberapa set karakter kemudian menentukan apakah *input* tersebut dapat diterima atau ditolak sesuai dengan pola yang terdefinisi oleh FA. Mesin atau pemodelan matematika ini cocok diterapkan pada komputer yang memiliki batasan memori tertentu. Contoh aplikasi *finite automata* dalam kehidupan sehari-hari adalah *vending machine*, pintu otomatis, dan pengatur lampu lalu lintas.

Mesin ini terdiri atas 5 elemen, yaitu himpunan *state* berhingga, *state* awal, set *final state*, *input*, dan aturan perpindahan *state*. Penggambarannya sendiri kurang lebih sebagai berikut :

$\{ Q, \Sigma, q, F, \delta \}$  dengan

$Q$  = himpunan set berhingga,

$\Sigma$  = himpunan simbol input,

$q$  = *state* awal,

$F$  = himpunan *state* akhir,

$\delta$  = aturan perpindahan *state* / fungsi transisi.

Terdapat dua jenis finite state/finite automata, yaitu deterministic finite automata (DFA) dan non-deterministic finite automata (NFA). DFA adalah mesin yang hanya dapat bergerak ke satu state tertentu dan tidak menerima *null* ( $\epsilon$ ) atau dalam kata lain tidak dapat berubah state tanpa input karakter. NFA adalah mesin yang dapat menerima *null* dan menerima berapapun jumlah state. Berikut adalah contoh penggunaan keduanya :

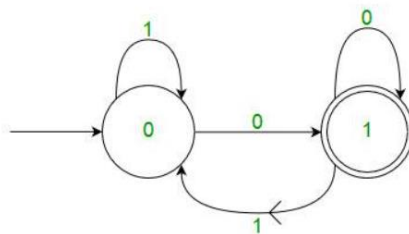
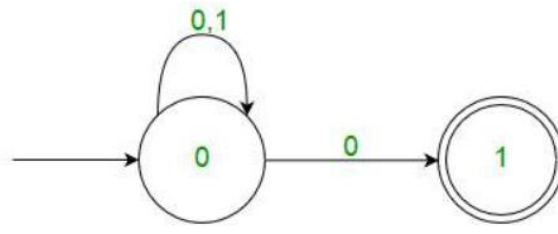


Figure: DFA with  $\Sigma = \{0, 1\}$



NFA

## 2. Context-free grammar

*Context-free grammar* merupakan tata bahasa formal yang aturan produksinya berbentuk :

$$A \rightarrow B, \text{ dengan}$$

A = pemproduksi (sebuah simbol nonterminal tunggal) dan

B = hasil produksi (dapat berupa simbol, terminal, variabel, maupun kosong).

Tata bahasa ini dapat didefinisikan sebagai 4 tuple, yaitu :

$$G = (V, \Sigma, R, S), \text{ dimana}$$

G = *grammar*

V = himpunan simbol non-terminal berhingga (biasa disebut *variables*)

$\Sigma$  = himpunan simbol terminal berhingga (*alphabet CFG*)

R = himpunan aturan produksi

S = simbol start (harus berupa nonterminal)

Pada CFG, simbol start digunakan untuk menurunkan suatu *string* secara berulang kali dengan mengganti non-terminal pada bagian hasil produksi hingga semua non-terminal berubah menjadi simbol terminal. Proses penyederhanaan CFG ini dilakukan untuk menghilangkan kerumitan dan aturan produksi yang tidak berarti. Terdapat tiga langkah dalam menyederhanakan CFG, eliminasi  $\epsilon$ -production, eliminasi unit production, dan eliminasi useless symbol. Berikut adalah contoh penyederhanaan dari CFG :

$$A \rightarrow cAB \mid ab$$

$$B \rightarrow BaC \mid C$$

$$C \rightarrow bC \mid \epsilon$$

Dengan mengeliminasi  $\epsilon$ , maka bentuknya akan menjadi sebagai berikut :

$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid C \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

Kemudian, setelah mengeliminasi *null*, dilakukan eliminasi *unit production* dimana apabila ada hasil produksi yang hanya mengandung 1 *variable*, maka hasil produksi itu akan diganti dengan

hasil produksi dari grammar dengan posisi *variable* menjadi pemproduksi. Oleh karena itu, hasilnya akan menjadi sebagai berikut :

$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid bC \mid b \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

Langkah terakhir adalah mengeliminasi *useless symbol*. Pada tahap ini, dilakukan uji generate dan uji reachable. Pada uji generate, terdapat dua contoh kasus, yaitu :

- 1) Lolos uji : jika  $X \rightarrow YZ$ ,  $Y \rightarrow aa$ ,  $Z \rightarrow b$
- 2) Tidak lolos uji : jika  $X \rightarrow YZ$ ,  $Y \rightarrow aa$ ,  $Z \rightarrow ZZ$  (X dan Z tidak lolos)

Kemudian, pada uji reachable, dilihat apakah seluruh aturan produksi dapat mencapai start atau tidak. Jika  $S \rightarrow XY$  dengan S sebagai start, maka X dan Y dapat dikatakan lolos uji reachable. Pada tahap ini, semua aturan produksi lolos uji, oleh sebab itu, hasil akhirnya akan berbentuk seperti berikut :

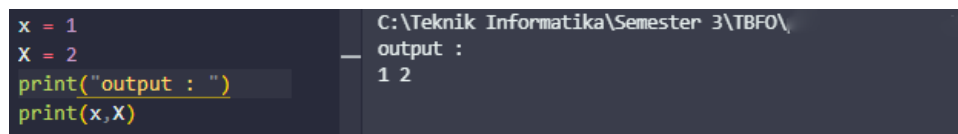
$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid bC \mid b \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

### 3. Syntax Python

Python merupakan bahasa yang *case-sensitive*, oleh sebab itu, penggunaan huruf kapital harus sangat diperhatikan karena akan menghasilkan makna yang berbeda. Fungsi dan variabel pada python penamaannya harus diperhatikan karena variabel x (dengan huruf kecil) dan variabel X (dengan huruf besar) memiliki arti yang berbeda,



```
x = 1
X = 2
print("output : ")
print(x,X)
```

C:\Teknik Informatika\Semester 3\TBF0\  
output :  
1 2

Fungsi, variabel, dan class harus dimulai oleh huruf (a-z) atau (A-Z), tidak boleh dimulai dengan angka ataupun karakter khusus. Selain itu, penamaan juga tidak boleh menggunakan reserved words seperti if, else, elif, return, break, import, dan lain sebagainya. Bahasa ini juga merupakan bahasa yang sensitif terhadap indentasi. Oleh sebab itu, ketika membuat suatu kode, segmentasinya harus sangat diperhatikan.

## BAB II

### HASIL

#### 1. FA

FA dibuat pada file FA.py dengan satu fungsi bernama FA yang menerima input berupa string dan mengeluarkan output berupa boolean. Berikut adalah penjelasan dari algoritma FA.

```
accepted = True
```

Pada awalnya, dibuat suatu boolean bernama accepted bernilai true untuk memulai proses pencarian kesalahan pada string.

```
if (len(s) == 0):  
    accepted = False
```

Kemudian, dibuat dua kondisi. Kondisi pertama adalah string kosong. Pada string kosong, accepted akan berubah nilai menjadi false karena tidak sesuai dengan syarat penamaan variabel.

```
else:  
    if not ((ord(s[0]) >= ord('A') and ord(s[0]) <= ord('Z')) or (ord(s[0]) >=  
ord('a') and ord(s[0]) <= ord('z')) or (s[0] == '_')):  
        accepted = False  
    else:  
        i = 1  
        while (i < len(s) and accepted):  
            if not ((ord(s[i]) >= ord('A') and ord(s[i]) <= ord('Z')) or (ord(s[i])  
>= ord('a') and ord(s[i]) <= ord('z')) or (s[i] == '_' or (ord(s[i]) >=  
ord('0') and ord(s[i]) <= ord('9')))):  
                accepted = False  
            else:  
                i += 1
```

Kondisi kedua menerima string yang memiliki isi (panjang string bukan 0). Pada awalnya, dilakukan pembacaan pada huruf pertama. Huruf pertama harus berupa letter atau \_. Apabila kondisi ini tidak terpenuhi, maka accepted akan berubah menjadi false karena tidak memenuhi kriteria. Namun, jika kondisi terpenuhi, maka dilakukan pengecekan dari index ke-1 hingga index terakhir dari string tersebut. Huruf-huruf tersebut harus berupa alfanumerik atau \_. Apabila kondisi terpenuhi, maka accepted tidak akan berubah. Namun, ketika kondisi tidak terpenuhi, maka accepted akan berubah menjadi false.

```
return accepted
```

Pada akhirnya, output yang akan dikeluarkan program adalah sebuah boolean true/false. Hasil ini bergantung pada nilai dari accepted yang sudah dicari sebelumnya.

## 2. CFG

Hasil dari CFG yang telah dibuat tersimpan dalam grammar.txt dan penjabarannya adalah sebagai berikut :

```
PROGRAM -> PROGRAM_STATEMENTS
PROGRAM_STATEMENTS -> ASSIGNMENT | FUNCTION_CALL | METHOD_CALL |
IF_STATEMENT | DEF_STATEMENT | CLASS_STATEMENT | IMPORT_STATEMENT |
RAISE_STATEMENT | WITH_STATEMENT | PASS_STATEMENT | FOR_STATEMENT |
WHILE_STATEMENT | COMMENT | COMMENT PROGRAM_STATEMENTS | _NEWLINE
PROGRAM_STATEMENTS | PROGRAM_STATEMENTS ; PROGRAM_STATEMENTS |
_NEWLINE
IDENTIFIER -> _IDENTIFIER | _IDENTIFIER . IDENTIFIER | LIST_OPER | DICT_OPER
ASSIGNMENT -> IDENTIFIER = DATA | IDENTIFIER _CPLUS DATA | IDENTIFIER _CMIN
DATA | IDENTIFIER _CDIVI DATA | IDENTIFIER _CDIVF DATA | IDENTIFIER _CMUL
DATA | IDENTIFIER _CPOW DATA | IDENTIFIER _CMOD DATA | ASSIGNMENT
SINGLINE_COMMENT | ASSIGNMENT PROGRAM_STATEMENTS
COMMENT -> SINGLINE_COMMENT | MULTLINE_COMMENT
MULTLINE_COMMENT -> _TRIPSQUOTE MULTLINE_COMMENT_CONTENT
_TRIPSQUOTE | _TRIPDQUOTE MULTLINE_COMMENT_CONTENT _TRIPDQUOTE
MULTLINE_COMMENT_CONTENT -> _ANY | _ANY MULTLINE_COMMENT_CONTENT |
INLINE_TERMINALS | INLINE_TERMINALS MULTLINE_COMMENT_CONTENT |
_NEWLINE | _NEWLINE MULTLINE_COMMENT_CONTENT
SINGLINE_COMMENT -> # SINGLINE_COMMENT_CONTENT
SINGLINE_COMMENT_CONTENT -> _ANY | _ANY SINGLINE_COMMENT_CONTENT |
INLINE_TERMINALS | INLINE_TERMINALS SINGLINE_COMMENT_CONTENT
WHILE_STATEMENT -> while DATA : _NEWLINE LOOP_BLOCK | while DATA :
COMMENT _NEWLINE LOOP_BLOCK
IFLOOP_STATEMENT -> if CONDITIONAL : _NEWLINE IFLOOP_BLOCK | if
CONDITIONAL : COMMENT _NEWLINE IFLOOP_BLOCK | IFLOOP_STATEMENT
_NEWLINE ELIFLOOP_STATEMENT | IFLOOP_STATEMENT _NEWLINE
ELSELOOP_STATEMENT
```

```
IFLOOP_BLOCK -> PROGRAM_STATEMENTS | ASSIGNMENT | FUNCTION_CALL |  
METHOD_CALL | IFLOOP_STATEMENT | DEF_STATEMENT | RAISE_STATEMENT |  
CLASS_STATEMENT | _NEWLINE IFLOOP_BLOCK | ASSIGNMENT IFLOOP_BLOCK |  
FUNCTION_CALL IFLOOP_BLOCK | METHOD_CALL IFLOOP_BLOCK |  
IFLOOP_STATEMENT IFLOOP_BLOCK | DEF_STATEMENT IFLOOP_BLOCK |  
RAISE_STATEMENT IFLOOP_BLOCK | CLASS_STATEMENT IFLOOP_BLOCK |  
WITH_STATEMENT | WITH_STATEMENT IFLOOP_BLOCK | IMPORT_STATEMENT |  
IMPORT_STATEMENT IFLOOP_BLOCK | PASS_STATEMENT | PASS_STATEMENT  
IFLOOP_BLOCK | LOOP_STATEMENT | LOOP_STATEMENT IFLOOP_BLOCK |  
FOR_STATEMENT | FOR_STATEMENT IFLOOP_BLOCK | WHILE_STATEMENT |  
WHILE_STATEMENT IFLOOP_BLOCK | COMMENT | COMMENT IFLOOP_BLOCK  
ELIFLOOP_STATEMENT -> elif CONDITIONAL : _NEWLINE IFLOOP_BLOCK | elif  
CONDITIONAL : COMMENT _NEWLINE IFLOOP_BLOCK | ELIFLOOP_STATEMENT  
ELIFLOOP_STATEMENT | ELIFLOOP_STATEMENT ELSELOOP_STATEMENT  
ELSELOOP_STATEMENT -> else : _NEWLINE IFLOOP_BLOCK | else : COMMENT  
_NEWLINE IFLOOP_BLOCK  
LOOP_STATEMENT -> continue | break | LOOP_STATEMENT COMMENT  
FOR_STATEMENT -> for IDENTIFIER in DATA : _NEWLINE LOOP_BLOCK | for  
IDENTIFIER in DATA : COMMENT _NEWLINE LOOP_BLOCK  
LOOP_BLOCK -> PROGRAM_STATEMENTS | _NEWLINE LOOP_BLOCK | ASSIGNMENT |  
ASSIGNMENT LOOP_BLOCK | FUNCTION_CALL | FUNCTION_CALL LOOP_BLOCK |  
METHOD_CALL | METHOD_CALL LOOP_BLOCK | IFLOOP_STATEMENT |  
IFLOOP_STATEMENT LOOP_BLOCK | DEF_STATEMENT | DEF_STATEMENT  
LOOP_BLOCK | RAISE_STATEMENT | RAISE_STATEMENT LOOP_BLOCK |  
PASS_STATEMENT | PASS_STATEMENT LOOP_BLOCK | LOOP_STATEMENT |  
LOOP_STATEMENT LOOP_BLOCK | FOR_STATEMENT | FOR_STATEMENT  
LOOP_BLOCK | WHILE_STATEMENT | WHILE_STATEMENT LOOP_BLOCK | COMMENT  
| COMMENT LOOP_BLOCK  
PASS_STATEMENT -> pass | PASS_STATEMENT COMMENT | PASS_STATEMENT  
PROGRAM_STATEMENTS  
WITH_STATEMENT -> with DATA as IDENTIFIER : _NEWLINE WITH_BLOCK | with  
DATA as IDENTIFIER : COMMENT _NEWLINE WITH_BLOCK  
WITH_BLOCK -> PROGRAM_STATEMENTS | _NEWLINE WITH_BLOCK | ASSIGNMENT |  
ASSIGNMENT WITH_BLOCK | FUNCTION_CALL | FUNCTION_CALL WITH_BLOCK |  
METHOD_CALL | METHOD_CALL WITH_BLOCK | IF_STATEMENT | IF_STATEMENT
```



WITH\_BLOCK | DEF\_STATEMENT | DEF\_STATEMENT WITH\_BLOCK |  
RAISE\_STATEMENT | RAISE\_STATEMENT WITH\_BLOCK | CLASS\_STATEMENT |  
CLASS\_STATEMENT WITH\_BLOCK | WITH\_STATEMENT | WITH\_STATEMENT  
WITH\_BLOCK | IMPORT\_STATEMENT | IMPORT\_STATEMENT WITH\_BLOCK |  
PASS\_STATEMENT | PASS\_STATEMENT WITH\_BLOCK | FOR\_STATEMENT |  
FOR\_STATEMENT WITH\_BLOCK | WHILE\_STATEMENT | WHILE\_STATEMENT  
WITH\_BLOCK | COMMENT | COMMENT WITH\_BLOCK  
RAISE\_STATEMENT -> raise | raise DATA | raise DATA from IDENTIFIER |  
RAISE\_STATEMENT COMMENT | RAISE\_STATEMENT PROGRAM\_STATEMENTS  
IMPORT\_STATEMENT -> import IDENTIFIER | import IDENTIFIER as IDENTIFIER | from  
IDENTIFIER import IDENTIFIER | from IDENTIFIER import \* | from IDENTIFIER import  
IDENTIFIER as IDENTIFIER | from IDENTIFIER import IMPORT\_IDENTIFIER |  
IMPORT\_STATEMENT COMMENT | IMPORT\_STATEMENT PROGRAM\_STATEMENTS  
IMPORT\_IDENTIFIER -> IDENTIFIER | IDENTIFIER , IMPORT\_IDENTIFIER  
CLASS\_STATEMENT -> class IDENTIFIER : \_NEWLINE CLASS\_BLOCK | class  
IDENTIFIER : COMMENT \_NEWLINE CLASS\_BLOCK  
CLASS\_BLOCK -> PROGRAM\_STATEMENTS | ASSIGNMENT | FUNCTION\_CALL |  
METHOD\_CALL | IF\_STATEMENT | DEF\_STATEMENT | RAISE\_STATEMENT |  
CLASS\_STATEMENT | \_NEWLINE CLASS\_BLOCK | ASSIGNMENT CLASS\_BLOCK |  
FUNCTION\_CALL CLASS\_BLOCK | METHOD\_CALL CLASS\_BLOCK | IF\_STATEMENT  
CLASS\_BLOCK | DEF\_STATEMENT CLASS\_BLOCK | RAISE\_STATEMENT  
CLASS\_BLOCK | CLASS\_STATEMENT CLASS\_BLOCK | WITH\_STATEMENT |  
WITH\_STATEMENT CLASS\_BLOCK | IMPORT\_STATEMENT | IMPORT\_STATEMENT  
CLASS\_BLOCK | PASS\_STATEMENT | PASS\_STATEMENT CLASS\_BLOCK |  
FOR\_STATEMENT | FOR\_STATEMENT CLASS\_BLOCK | WHILE\_STATEMENT |  
WHILE\_STATEMENT CLASS\_BLOCK | COMMENT | COMMENT CLASS\_BLOCK  
DEF\_STATEMENT -> def IDENTIFIER ( ) : \_NEWLINE DEF\_BLOCK | def IDENTIFIER  
( DEF\_PARAMS ) : \_NEWLINE DEF\_BLOCK | def IDENTIFIER ( ) : COMMENT \_NEWLINE  
DEF\_BLOCK | def IDENTIFIER ( DEF\_PARAMS ) : COMMENT \_NEWLINE DEF\_BLOCK  
DEF\_PARAMS -> IDENTIFIER | IDENTIFIER , DEF\_PARAMS  
DEF\_BLOCK -> PROGRAM\_STATEMENTS | ASSIGNMENT | FUNCTION\_CALL |  
METHOD\_CALL | IFRET\_STATEMENT | RETURN\_STATEMENT | DEF\_STATEMENT |  
RAISE\_STATEMENT | CLASS\_STATEMENT | \_NEWLINE DEF\_BLOCK | ASSIGNMENT  
DEF\_BLOCK | FUNCTION\_CALL DEF\_BLOCK | METHOD\_CALL DEF\_BLOCK |  
IFRET\_STATEMENT DEF\_BLOCK | RETURN\_STATEMENT DEF\_BLOCK |

```
DEF_STATEMENT DEF_BLOCK | RAISE_STATEMENT DEF_BLOCK |  
CLASS_STATEMENT DEF_BLOCK | WITH_STATEMENT | WITH_STATEMENT  
DEF_BLOCK | IMPORT_STATEMENT | IMPORT_STATEMENT DEF_BLOCK |  
PASS_STATEMENT | PASS_STATEMENT DEF_BLOCK | FOR_STATEMENT |  
FOR_STATEMENT DEF_BLOCK | WHILE_STATEMENT | WHILE_STATEMENT  
DEF_BLOCK | COMMENT | COMMENT DEF_BLOCK  
RETURN_STATEMENT -> return DATA | RETURN_STATEMENT COMMENT  
IF_STATEMENT -> if CONDITIONAL : _NEWLINE IF_BLOCK | if CONDITIONAL :  
COMMENT _NEWLINE IF_BLOCK | IF_STATEMENT _NEWLINE ELIF_STATEMENT |  
IF_STATEMENT _NEWLINE ELSE_STATEMENT  
IF_BLOCK -> PROGRAM_STATEMENTS | ASSIGNMENT | FUNCTION_CALL |  
METHOD_CALL | IF_STATEMENT | DEF_STATEMENT | RAISE_STATEMENT |  
CLASS_STATEMENT | _NEWLINE IF_BLOCK | ASSIGNMENT IF_BLOCK |  
FUNCTION_CALL IF_BLOCK | METHOD_CALL IF_BLOCK | IF_STATEMENT IF_BLOCK |  
DEF_STATEMENT IF_BLOCK | RAISE_STATEMENT IF_BLOCK | CLASS_STATEMENT  
IF_BLOCK | WITH_STATEMENT | WITH_STATEMENT IF_BLOCK |  
IMPORT_STATEMENT | IMPORT_STATEMENT IF_BLOCK | PASS_STATEMENT |  
PASS_STATEMENT IF_BLOCK | FOR_STATEMENT | FOR_STATEMENT IF_BLOCK |  
WHILE_STATEMENT | WHILE_STATEMENT IF_BLOCK | COMMENT IF_BLOCK  
ELIF_STATEMENT -> elif CONDITIONAL : _NEWLINE IF_BLOCK | elif CONDITIONAL :  
COMMENT _NEWLINE IF_BLOCK | ELIF_STATEMENT _NEWLINE ELIF_STATEMENT |  
ELIF_STATEMENT _NEWLINE ELSE_STATEMENT  
ELSE_STATEMENT -> else : _NEWLINE IF_BLOCK | else : COMMENT _NEWLINE  
IF_BLOCK  
IFRET_STATEMENT -> if CONDITIONAL : _NEWLINE IFRET_BLOCK | if  
CONDITIONAL : COMMENT _NEWLINE IFRET_BLOCK | IFRET_STATEMENT  
_NEWLINE ELIFRET_STATEMENT | IFRET_STATEMENT _NEWLINE  
ELSERET_STATEMENT  
IFRET_BLOCK -> PROGRAM_STATEMENTS | ASSIGNMENT | FUNCTION_CALL |  
METHOD_CALL | IFRET_STATEMENT | DEF_STATEMENT | RAISE_STATEMENT |  
CLASS_STATEMENT | RETURN_STATEMENT | _NEWLINE IFRET_BLOCK |  
ASSIGNMENT IFRET_BLOCK | FUNCTION_CALL IFRET_BLOCK | METHOD_CALL  
IFRET_BLOCK | IFRET_STATEMENT IFRET_BLOCK | DEF_STATEMENT IFRET_BLOCK |  
RAISE_STATEMENT IFRET_BLOCK | CLASS_STATEMENT IFRET_BLOCK |  
RETURN_STATEMENT IFRET_BLOCK | WITH_STATEMENT | WITH_STATEMENT
```

```
IFRET_BLOCK | IMPORT_STATEMENT | IMPORT_STATEMENT IFRET_BLOCK |  
PASS_STATEMENT | PASS_STATEMENT IFRET_BLOCK | FOR_STATEMENT |  
FOR_STATEMENT IFRET_BLOCK | WHILE_STATEMENT | WHILE_STATEMENT  
IFRET_BLOCK | COMMENT | COMMENT IFRET_BLOCK  
ELIFRET_STATEMENT -> elif CONDITIONAL : _NEWLINE IFRET_BLOCK | elif  
CONDITIONAL : COMMENT _NEWLINE IFRET_BLOCK | ELIFRET_STATEMENT  
_NEWLINE ELIFRET_STATEMENT | ELIFRET_STATEMENT _NEWLINE  
ELSERET_STATEMENT  
ELSERET_STATEMENT -> else : _NEWLINE IFRET_BLOCK | else : COMMENT _NEWLINE  
IFRET_BLOCK  
DATA -> NONTERNARY_DATA | TERNARY_DATA  
NONTERNARY_DATA -> STRING | _INTEGER | FLOAT | LIST | TUPLE | SET |  
DICTIONARY | BOOLEAN | IDENTIFIER | CONDITIONAL | MATH_OPERATION |  
FUNCTION_CALL | METHOD_CALL | None | LIST_OPER | DICT_OPER | LIST_SLICE  
TERNARY_DATA -> DATA if CONDITIONAL else DATA  
DICT_IDENTIFIER -> DICTIONARY | IDENTIFIER  
DICT_OPER -> DICT_IDENTIFIER [ DATA ]  
LIST_IDENTIFIER -> LIST | TUPLE | IDENTIFIER | FUNCTION_CALL | METHOD_CALL |  
STRING  
LIST_OPER -> LIST_IDENTIFIER [ DATA ]  
LIST_SLICE -> LIST_IDENTIFIER [ DATA : DATA ] | LIST_IDENTIFIER [ DATA : ] |  
LIST_IDENTIFIER [ : DATA ] | LIST_IDENTIFIER [ : ] | LIST_IDENTIFIER [ DATA : DATA :  
DATA ] | LIST_IDENTIFIER [ DATA : DATA : ] | LIST_IDENTIFIER [ DATA : : DATA ] |  
LIST_IDENTIFIER [ : DATA : DATA ] | LIST_IDENTIFIER [ DATA : : ] | LIST_IDENTIFIER  
[ : DATA : ] | LIST_IDENTIFIER [ : : DATA ] | IDENTIFIER [ : : ]  
LIST -> [ ] | [ LIST_DATA ]  
LIST_DATA -> DATA | DATA , LIST_DATA | DATA for IDENTIFIER in DATA  
FLOAT -> _INTEGER . _INTEGER . _INTEGER  
STRING -> " ANY " | ' ANY ' | " " | ''  
ANY -> _ANY | _ANY ANY | INLINE_TERMINALS | INLINE_TERMINALS ANY  
TUPLE -> ( ) | ( DATA , ) | ( DATA , TUPLE_DATA )  
TUPLE_DATA -> DATA | DATA , TUPLE_DATA  
SET -> { SET_DATA }  
SET_DATA -> DATA | DATA , SET_DATA  
DICTIONARY -> { } | { DICT_DATA }
```

DICT\_DATA -> STRING : DATA | STRING : DATA , DICT\_DATA  
BOOLEAN -> True | False  
CONDITIONAL -> DATA | DATA \_EQ DATA | DATA \_NEQ DATA | DATA \_GE DATA |  
DATA \_LE DATA | DATA > DATA | DATA < DATA | DATA in DATA | DATA is DATA | not  
CONDITIONAL | CONDITIONAL or CONDITIONAL | CONDITIONAL and CONDITIONAL |  
( CONDITIONAL )  
MATH\_OPERATION -> DATA + DATA | DATA - DATA | DATA \* DATA | DATA / DATA |  
DATA \_DIV DATA | DATA \_POW DATA | DATA % DATA | ( MATH\_OPERATION ) |  
MATH\_OPERATION MATH\_OPERATION  
FUNCTION\_CALL -> IDENTIFIER ( ) | IDENTIFIER ( FUNC\_PARAMS ) | FUNCTION\_CALL  
PROGRAM\_STATEMENTS  
FUNC\_PARAMS -> DATA | DATA , FUNC\_PARAMS  
METHOD\_IDENTIFIER -> LIST\_IDENTIFIER | DICT\_IDENTIFIER  
METHOD\_CALL -> METHOD\_IDENTIFIER . IDENTIFIER ( ) | METHOD\_IDENTIFIER .  
IDENTIFIER ( METHOD\_PARAMS ) | METHOD\_CALL PROGRAM\_STATEMENTS  
METHOD\_PARAMS -> DATA | DATA , METHOD\_PARAMS  
INLINE\_TERMINALS -> \_ANY | \_INTEGER | \_IDENTIFIER | \_EQ | \_NEQ | \_GE | \_LE | \_DIV |  
\_CPLUS | \_CMIN | \_CDIVI | \_CDIVF | \_CMUL | \_POW | \_CPOW | > | < | not | and | or | True |  
False | ' | " | [ | ] | { | } | ( | ) | : | , | . | = | + | - | / | \* | : | if | else | elif | def | return | None | class | in | is |  
from | import | as | raise | with | pass | continue | break | for | while | ; | # | \_TRIPSQUOTE |  
\_TRIPDQUOTE | % | \_CMOD

## BAB III

### IMPLEMENTASI DAN PENGUJIAN

#### 1. Spesifikasi Teknis Program

Struktur data dari compiler bahasa Python adalah sebagai berikut :

```
pythonParser/  
├─ CNF.py  
├─ FA.py  
├─ README.md  
├─ generatedCNF.txt  
├─ grammar.txt  
├─ inputAcc.py  
├─ inputReject.py  
├─ lab00_lovemeter.py  
├─ main.py  
├─ parser.py  
├─ pythonGrammar.txt  
├─ simulateCYK.py  
├─ tactical_doll.py  
├─ test1.py  
├─ test2.py  
├─ test3.py
```

Terdapat 4 file python utama yang digunakan untuk membuat algoritma, antara lain :

a) parser.py

File ini berisi algoritma-algoritma parser yang digunakan untuk melakukan kompilasi. Fungsi dan prosedur yang terdapat pada file ini adalah sebagai berikut :

<b>function</b> generateGrammar (filename : string) → array of array of string { Membuat dictionary grammar dari suatu file yang berisi string }
<b>procedure</b> cyk ( <u>input</u> grammar : array of array of string, <u>input</u> tokens : array of string) { I.S. Tokens adalah sebuah array of string dan grammar adalah sebuah dictionary. Keduanya terdefinisi }

{ F.S. Menentukan apakah barisan token 'tokens' memenuhi grammar atau tidak }
---

<b><u>function</u></b> convertLine (l : string) → array of string { Mengkonversi sebuah line yang dibaca, 'l' menjadi list of tokens. }
--

b) CNF.py

File ini berisi algoritma-algoritma yang berfungsi untuk mengubah suatu txt file berisi CFG menjadi CNF. Hasil dari konversi akan disimpan dalam bentuk txt file.

<b><u>function</u></b> generateCFG (filename : string) → array of array of string { Menghasilkan CFG dari suatu file yang berisi string }
--

<b><u>procedure</u></b> removeUnitProduction ( <u>input/output</u> R : array of array of string) { I.S. R adalah CFG yang sudah terdefinisi dan tidak kosong } { F.S. Menghapus atau mensubstitusi hasil produksi yang hanya terdiri dari satu variabel dengan hasil produksi dari grammar dengan posisi variabel menjadi pemproduksi }
---

<b><u>function</u></b> getAvailableVariableName (name : string, R : array of array of string) → string { Mengambil nama variabel yang tersedia pada R }
--

<b><u>procedure</u></b> simplifyRule ( <u>input/output</u> R : array of array of string) { I.S. R adalah CFG yang sudah terdefinisi dan tidak kosong } { F.S. Mengkonversi R menjadi bentuk yang lebih sederhana }
--

<b><u>procedure</u></b> writeRule ( <u>input/output</u> filename : string, <u>input</u> R : array of array of string) { I.S. Nama file (filename) dan R sudah terdefinisi } { F.S. Hasil akhir R berbentuk CNF tersimpan dalam suatu filename.txt }
---

c) main.py

Pada file ini, seluruh algoritma-algoritma dari file lain dikumpulkan dan dipanggil untuk melakukan kompilasi bahasa python. Hanya terdapat satu prosedur pada file ini, yaitu prosedur untuk mencetak logo.

<b><u>procedure</u></b> logo () { F.S. Menuliskan welcome python ke layar }
--

d) FA.py

File ini berisi algoritma yang digunakan untuk mendeteksi nama variabel menggunakan fungsi FA. Fungsi yang terdapat pada file ini adalah :

```
function FA (s : string) → boolean  
{ Mengemulasikan suatu FA untuk mengetahui apakah nama variabel  
benar/salah. Mengembalikan true jika nama variabel valid, dan false jika  
tidak. }
```

## 2. Uji Kasus

<i>Uji kasus #1 (inputAcc.py)</i>
<pre>def do_something(x):     ''' This is a sample multiline comment     ...      if x == 0:         return 0     elif x + 4 == 1:         if True:             return 3         else:             return 2     elif x == 32:         return 4     else:         return "Doodoo"</pre>
Menghasilkan hasil VERDICT: Accepted

```
PS C:\Programming\pythonParser> python main.py inputAcc.py
```

### Uji kasus #2 (*inputReject.py*)

```
def do_something(x):
    ''' This is a sample multiline comment
    ...

    x + 2 = 3

    if x == 0 + 1:
        return 0

    elif x + 4 == 1:
        else:
            return 2

    elif x == 32:
        return 4

    else:
        return "Doodoo"
```

Menghasilkan hasil VERDICT: Syntax Error



```
PS C:\Programming\pythonParser> python main.py inputReject.py
```

WELCOME

# PYTHON PARSER

=====

Ketik X apabila anda ingin keluar dari program!

=====

Processing inputReject.py!  
VERDICT: Syntax Error

### *Uji kasus #3 (test1.py)*

Menguji class, class access, list comprehension, list access, loop, dan with statement.

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand
        self.color = color

listOfCars = [None for i in range(10)]

with open('cars.txt') as f:
    lines = f.readlines()
    for i in range(10):
        arr = lines[i].split(" ")
        carToAdd = Car(lines[0], lines[1])
        listOfCars[i] = carToAdd
```

---

Menghasilkan VERDICT: Accepted

```
PS C:\Programming\pythonParser> python main.py test1.py

          _ _ _ _ _
         / / / \ \ \ \
        / / / \ \ \ \
       / / / \ \ \ \
      / / / \ \ \ \
     / / / \ \ \ \
    / / / \ \ \ \
   / / / \ \ \ \
  / / / \ \ \ \
 / / / \ \ \ \
/ / / \ \ \ \

PYTHON PARSE

=====
Ketik X apabila anda ingin keluar dari program!
=====
Processing test1.py!
VERDICT: Accepted
```

#### *Uji kasus #4 (test2.py)*

Menguji multi-line comment, pass statement, single-line comment, pemanggilan fungsi, serta pengecekan quotation mark.

```
'''
This is a multi-line comment.
'''

pass

# This is a single-line comment.

print("This string has matching quotations.")
print("This string has unmatched quotations.")
```

Menghasilkan hasil VERDICT: Syntax Error

```
PS C:\Programming\pythonParser> python main.py test2.py

          _ _ _ _ _
         / / / \ \ \ \
        / / / \ \ \ \
       / / / \ \ \ \
      / / / \ \ \ \
     / / / \ \ \ \
    / / / \ \ \ \
   / / / \ \ \ \
  / / / \ \ \ \
 / / / \ \ \ \
/ / / \ \ \ \

PYTHON PARSE

=====
Ketik X apabila anda ingin keluar dari program!
=====
Processing test2.py!
VERDICT: Syntax Error
```

***Uji kasus #5 (lab00\_lovemeter.py)***

Menguji import, pemanggilan method, fungsi dengan banyak argumen, perbandingan. Diambil dari lab CSUI semester 1.

```
import random
print("SELAMAT DATANG DI LOVEMETER")

nama_dia = input("Nama calon jodoh: ")

cocok = random.random()
print("Kecocokan anda", cocok*100, "%")

if cocok > 0.8:
    print("Anda sangat cocok dengan " + nama_dia + "!")
elif 0.5 <= cocok <= 0.8:
    print("Anda lumayan cocok dengan " + nama_dia + "!")
else:
    print("Anda tidak cocok dengan " + nama_dia + "!")
```

Menghasilkan VERDICT: Accepted

PS C:\Programming\pythonParser> python main.py lab00\_lovemeter.py



```
=====
                          Ketik X apabila anda ingin keluar dari program!
=====
Processing lab00_lovemeter.py!
VERDICT: Accepted
```

***Uji kasus #6 (tactical\_doll.py)***

Menguji nested math operation. Diambil dari lab CSUI semester 1.

```
Nama_TD = input("Siapakah nama Tactical Dollmu? ")
Firepower = int(input("Besarnya kekuatan firepower = "))
RateOfFire = int(input("Besarnya kekuatan rate of fire = "))
Accuracy = int(input("Besarnya akurasi = "))
```

```
Evasion = int(input("Besar kemampuan evasion = "))

DamagePerSecond = (Firepower+RateOfFire)/60
CombatEffectiveness = (30 * Firepower + 40 * ((RateOfFire**2)/120)) + 15
* (Accuracy + Evasion)

print()
print("Nama Tactical Dollmu adalah " + Nama_TD)
print("Besar kekuatan fire power " + Nama_TD + " adalah " +
str(Firepower))
print("Besar kekuatan rate of fire " + Nama_TD + " adalah " +
str(RateOfFire))
print("Besar accuracy " + Nama_TD + " adalah " + str(Accuracy))
print("Besar kemampuan evasion " + Nama_TD + " adalah " + str(Evasion))

print()
print("Damage per second " + Nama_TD + " adalah " +
str(round(DamagePerSecond,2)))
print("Combat effectiveness " + Nama_TD + " adalah " +
str(round(CombatEffectiveness)))
```

Menghasilkan VERDICT: Accepted



```
PS C:\Programming\pythonParser> python main.py tactical_doll.py

          _ _ _ _ _
         / / / / /
        / / / / /
       / / / / /
      / / / / /
     / / / / /
    / / / / /
   / / / / /
  / / / / /
 / / / / /
/ / / / /

PYTHON PARSER

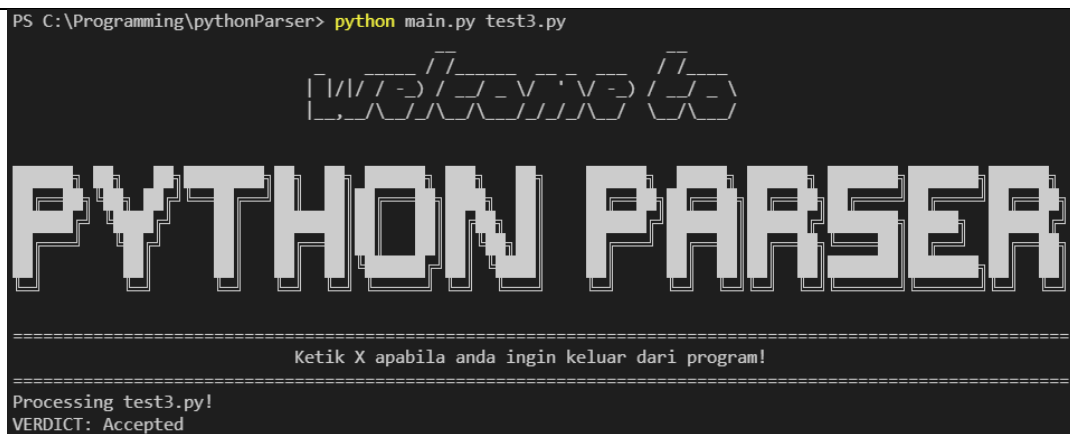
=====
Ketik X apabila anda ingin keluar dari program!
=====
Processing tactical_doll.py!
VERDICT: Accepted
```

*Uji kasus #7 (test3.py)*

Menguji .format dan akses array multidimensi

```
jumlah_operasi = int(input("Masukkan banyak operasi: "))  
for i in range(jumlah_operasi):  
    operasi = input("Masukkan operasi: ")  
    list_operasi = operasi.split()  
    if 'BELI' == list_operasi[0]:  
        beli_keranjang(list_operasi[1], list_operasi[2])  
        print("Berhasil menambahkan {} dengan kapasitas  
{}}".format(daftar_keranjang[0][0], daftar_keranjang[0][1]))
```

Menghasilkan VERDICT: Accepted



```
PS C:\Programming\pythonParser> python main.py test3.py  
  
          _ _ _ _ _  
      _ _ / _ _ _ _ \ _ _  
     / _ / _ _ _ _ \ _ /  
    / _ / _ _ _ _ \ _ /  
   / _ / _ _ _ _ \ _ /  
  / _ / _ _ _ _ \ _ /  
 / _ / _ _ _ _ \ _ /  
/ _ / _ _ _ _ \ _ /  
  
PYTHON PARSE  
  
===== Ketik X apabila anda ingin keluar dari program! =====  
Processing test3.py!  
VERDICT: Accepted
```

## **BAB IV**

### **PEMBAGIAN TUGAS**

<b>No</b>	<b>Nama</b>	<b>Tugas</b>
1	Marchotridyo	Membuat rules + cfg.txt + parser
2	Januar Budi Ghifari	Membuat CFG to CNF
3	Rania Dwi Fadhillah	Membuat main

## **LAMPIRAN**

### **a. Github Repository**

<https://github.com/acomarcho/pythonParser.git>

