

IF2124 TEORI BAHASA FORMAL DAN OTOMATA

LAPORAN TUGAS BESAR **Compiler Bahasa Python**



Oleh:

Kelompok seadanya-sebisanya

Fransiskus Davin Anwari	13520025
Taufan Fajarama Putrawansyah R	13520031
Tri Sulton Adila	13520033

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021

DAFTAR ISI

DAFTAR ISI.....	i
BAB 1 Teori Dasar.....	1
1.1. Finite Automata.....	1
1.1.1. Diagram Transisi FA.....	1
1.1.2. Tabel Transisi FA	2
1.2. Context-Free Grammar	2
1.2.1. Parsing dengan CFG	2
1.3. Chomsky Normal Form	3
1.3.1. Mengubah CFG ke dalam CNF	4
1.4. Algoritma Cocke-Younger-Kasami	5
1.5. Sintak Python	5
BAB 2 Hasil FA dan CFG	8
2.1. Pembuatan Finite Automata	8
2.2. Pembuatan Context-Free Grammar	8
2.3. Hasil Konversi CFG ke dalam CNF	10
BAB 3 Implementasi dan Pengujian	14
3.1. Spesifikasi Program.....	14
3.1.1. File CFG2CNF.py	14
3.1.2. File main.py	15
3.1.3. File cyk.py.....	15
3.1.4. File tokenizationVar.py.....	16
3.1.5. File variableChecking.py	16
3.2. Hasil Pengujian.....	16
BAB 4 Penutup.....	20
4.1. Kesimpulan.....	20
4.2. Saran	20
4.3. Link Repository Github	20
4.4. Pembagian Tugas.....	20
DAFTAR REFERENSI.....	21

BAB 1

Teori Dasar

1.1. Finite Automata

Finite Automata (FA) adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa reguler. Finite automata dapat diimplementasikan dengan sejumlah konfigurasi internal berupa state. Contoh penerapannya pada sistem adalah mesin jaja dan lexical analyser. Definisi formal FA adalah list dari lima komponen, yaitu kumpulan state, input, aturan perpindahan, state awal, dan state akhir. FA secara formal dapat ditulis dari lima komponen $(Q, \Sigma, \delta, q_0, F)$.

1. Q adalah himpunan set berhingga yang disebut sebagai himpunan states
2. Σ adalah himpunan berhingga alfabet dari simbol.
3. $\delta : Q \times \Sigma$ adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state.
4. $q_0 \in Q$ adalah state awal.
5. $F \subseteq Q$ adalah himpunan states akhir.

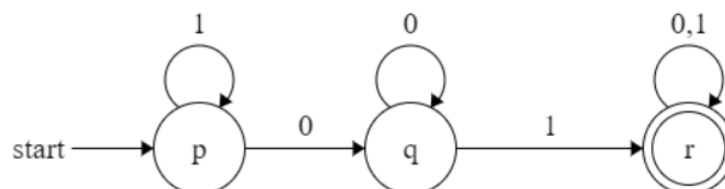
Apabila suatu string membawa state FA dari state inisial ke salah satu state dalam F dan berakhir pada state tersebut, string tersebut diterima sebagai anggota bahasa tersebut. Finite automata dapat disajikan dengan beberapa cara, yaitu dengan diagram transisi yang berupa suatu graf dan tabel transisi yang berupa daftar berbentuk tabel untuk fungsi δ , yang merupakan hubungan antara himpunan states dengan alfabet input.

1.1.1. Diagram Transisi FA

Diagram transisi untuk FA $(Q, \Sigma, \delta, q_0, F)$ adalah sebagai berikut.

1. Terdapat simpul untuk setiap states pada Q .
2. Untuk setiap $\delta(q, a) = p$ akan diwakili dengan busur berlabel a dari state q ke state p .
3. State awal q_0 akan ditunjuk oleh tanda panah berlabel start yang tidak berasal dari state manapun.
4. State akhir (F) adalah state yang memiliki lingkaran ganda.

Berikut merupakan contoh dari diagram transisi dengan state awal adalah p dan F adalah r .



1.1.2. Tabel Transisi FA

Tabel transisi merupakan tabel yang merepresentasikan fungsi δ yang mengambil dua argumen dan menghasilkan suatu nilai. State akan berkorespondensi dengan baris pada tabel, sedangkan input akan berkorespondensi dengan kolom pada tabel. Berikut merupakan contoh tabel transisi FA dari diagram transisi di atas.

δ	0	1
\rightarrow p	q	p
q	q	r
* r	r	r

Tabel transisi di atas memiliki arti sebagai berikut.

1. Simbol pada baris paling atas adalah input.
2. Simbol pada kolom paling kiri adalah state.
3. Simbol yang berada di dalam tabel adalah fungsi transisi.
4. Simbol anak panah menunjukkan state tersebut adalah state awal (start).
5. Simbol bintang (*) menunjukkan state tersebut adalah state akhir (final).

1.2. Context-Free Grammar

Context-Free Grammar adalah sebuah tata bahasa yang tidak mempunyai pembatasan pada hasil produksinya. Misalnya dengan tata bahasa yang berbeda terdapat peraturan dimana pada ruas kiri hanyalah sebuah simbol variabel.

$$\alpha \rightarrow \beta$$

Sedangkan pada CFG tidak ada batasan ruas kiri suatu bahasa dapat menunjukkan arti apa sebanyak apapun, misalnya contohnya sebagai berikut:

$$B \rightarrow CDeFg$$

$$D \rightarrow BcDe$$

Dimana ruas kiri bahasa tersebut (B dan D) dapat menunjukan lebih dari satu simbol variabel dan terminal. Definisi formal dari CFG dapat dituliskan menjadi

$$G = (V, T, P, S)$$

V = Himpunan terbatas variabel

T = Himpunan terbatas terminal

P = Himpunan terbatas dari produksi

S = Start simbol

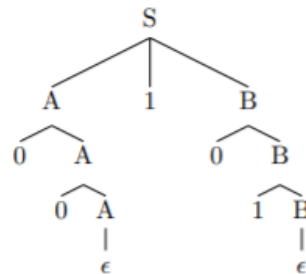
Pada umumnya *Context-Free Grammar* memiliki tujuan yang sama seperti tata bahasa reguler lainnya, yaitu untuk menunjukan bagaimana menghasilkan suatu untai-untai pada suatu bahasa, hanya saja seperti dengan namanya CFG memiliki kebebasan yang lebih besar dalam perancangannya karena tidak ada batasan pada hasil produksi.

1.2.1. Parsing dengan CFG

Salah satu kegunaan adalah sebagai dasar dari pembentuk suatu proses analisis sintaksis. Bagian-bagian sintaks pada suatu kompilator dapat kita definisikan di dalam

CFG. Untuk menggambar simbol-simbol variabel kita juga dapat menggunakan pohon penurunan agar dapat menjadi simbol-simbol terminal.

Contohnya pada suatu bahasa yang menggambarkan ekspresi reguler $0^*1(0+1)^*$ kita dapat membuat pohon penurunan untuk mencari string 00101 sebagai berikut



Jika kita perhatikan pohon penurunan tersebut mempunyai 3 cabang yaitu ke kiri ke kanan dan ketengah, pada bahasa ini kita dapat memulai penurunan dari kiri atau kanan, dimana kita akan memperluas variabel terkiri atau terkanan pada string yang kita inginkan, pada konteks ini variabel terkiri adalah 0 dan variabel terkanan adalah 1.

Untuk memperoleh string 00101 maka kita perlu mencari turunan terkirinya yaitu

$$S \rightarrow A1B \rightarrow 0A1B \rightarrow 00A1B \rightarrow 001B \rightarrow 0010B \rightarrow 00101B \rightarrow 00101$$

Sedangkan jika kita mencari turunan terkanannya didapat

$$S \rightarrow A1B \rightarrow A10B \rightarrow A101B \rightarrow A101 \rightarrow 0A101 \rightarrow 00A101 \rightarrow 00101$$

Perhatikan bahwa dengan mencari turunan dari terkiri dan terkanan kita pada akhirnya akan tetap mendapat hasil yang sama, karena itu bisa dibilang bahwa pohon keturunan ini fleksibel penggunaannya karena akan menghasilkan hasil yang sama.

1.3. Chomsky Normal Form

Chomsky Normal Form atau disingkat CNF merupakan salah satu bentuk normal yang dapat digunakan untuk mengolah CFG. Dapat dibilang bahwa CNF adalah hasil olehan CFG yang sudah mengalami penyederhanaan dengan menghilangkan produksi useless, unit, dan epsilon (ϵ). Karena itu kita juga langsung bisa menyimpulkan bahwa semua CFG yang tidak memiliki produksi useless, unit dan ϵ juga termasuk CNF. Maka semua produksi CFG yang sudah berbentuk CNF akan memiliki produksi antara 2 contoh berikut :

$$A \rightarrow AB \text{ atau } A \rightarrow a$$

Jika kita ingin membaca suatu kode kemudian mengcompile kode tersebut maka salah satu teknik yang perlu digunakan adalah mengubah CFG kode tersebut menjadi CNF

1.3.1. Mengubah CFG ke dalam CNF

Jika ingin mengubah suatu CFG ke CNF maka kita perlu menyimak kembali peraturan dari CNF, yaitu tidak ada produksi useless, unit, dan epsilon. Maka hal pertama yang kita perlu lakukan adalah menghilangkan produksi useless, unit, dan epsilon pada CFG yang bersangkutan.

Setelah itu kita perlu juga memisah produksi yang sudah sesuai pada CNF dan yang belum sesuai. Contohnya jika ada suatu aturan produksi yang membentuk terminal bersebelahan dengan variabel, maka terminal kita perlu ubah agar yang bersebelahan adalah sesama variabel.

$$S \rightarrow aB \Rightarrow S \rightarrow P1B, P1 \rightarrow a$$

Seperti pada contoh di atas, ketika suatu produksi yang memuat terminal dan variabel dipisah, terminal diubah menjadi suatu variabel yang nantinya hanya memproduksi terminal itu saja. Hingga akan terbentuk aturan produksi baru untuk setiap terminal. Untuk mempermudah pengertian diberikan contoh berikut sebagai pengubah CFG menjadi CNF

CFG :

$$S \rightarrow aA \mid AB$$

$$A \rightarrow a \mid bc$$

$$B \rightarrow AC \mid Bb$$

$$C \rightarrow A \mid c$$

Pemisahan aturan produksi yang valid di CNF dan tidak:

CNF Valid:

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow AC$$

$$C \rightarrow A \mid c$$

Not CNF Valid:

$$S \rightarrow aA$$

$$A \rightarrow bc$$

$$B \rightarrow Bb$$

Perubahan aturan produksi yang non valid dengan penambahan aturan produksi terminal:

$$S \rightarrow aA \quad \Rightarrow S \quad \rightarrow X1A$$

$$A \rightarrow bc \quad \Rightarrow A \quad \rightarrow X2X3$$

$$B \rightarrow Bb \quad \Rightarrow B \quad \rightarrow BX2$$

Aturan produksi yang terbentuk ditambah ke CNF:

$$X1 \rightarrow a$$

$$X2 \rightarrow b$$

$$X3 \rightarrow c$$

Hasil akhir:

$$S \rightarrow AB \mid X1AX1 \quad \rightarrow a$$

$$A \rightarrow X1 \mid X2X3X2 \rightarrow b$$

$$B \rightarrow AC \mid BX2X3 \quad \rightarrow c$$

$$C \rightarrow A \mid X3$$

Setelah dilakukan semua hal seperti yang diatas maka CFG akan sudah berbentuk CNF sehingga sudah dapat kita gunakan untuk mengecek Sintaks suatu kode yang akan kita kompilasi.

1.4. Algoritma Cocke-Younger-Kasami

Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma parsing untuk menentukan apakah suatu string dapat diterima oleh suatu Bahasa bebas-konteks (Context-free grammar/CFG). CFG yang diterima oleh algoritma CYK adalah CFG dalam bentuk Chomsky Normal Form (CNF). Proses parsing dengan algoritma CYK memanfaatkan struktur data sebuah array dua dimensi dan merupakan aplikasi *dynamic programming* (Program Dinamis) karena proses parsing memanfaatkan hasil parsing sebelumnya untuk memutuskan apakah proses yang sedang berlangsung dapat diterima maupun tidak.

Misalkan suatu CFG didefinisikan sebagai

$$G = (\{S,A,B,C\}, \{a,b\}, P, S)$$

dan mempunyai production rule:

$$S \rightarrow AB \mid BC$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow AB \mid a$$

S,A,C				
-	S,C,A			
-	B	B		
A,S	B	S,C	A,S	
B	A,C	A,C	B	A,C
	b	a	a	b
				a

String 'baaba' diterima oleh CFG tersebut karna dapat mencapai startsymbol S setelah dilakukan iterasi *bottom-up* dengan pseudocode dari algoritma CYK di bawah ini.

Algoritma CYK

```

begin
1. for i := 1 to n do
2.  $V_{i1} := \{A | A \rightarrow a \text{ aturan produksi dimana}$ 
   simbol ke i adalah a};
3. for j := 2 to n do
4. for i := 1 to (n - j + 1) do
   begin
 $V_{ij} := \emptyset$ ;
   for k:= 1 to (j-1) do
 $V_{ij} := V_{ij} \cup \{A | A \rightarrow BC \text{ adalah suatu}$ 
   produksi, dimana B di  $V_{ik}$  dan C di
 $V_{i+k,j-k}\}$ 
   end
   end
end

```

1.5. Sintak Python

Sintak merupakan kumpulan aturan yang mendefinisikan suatu bentuk bahasa. Sintak berfungsi untuk membentuk suatu kalimat sebagai barisan atau urutan dari pemilihan suatu kata dasar. Kata dapat dibentuk oleh karakter-karakter alfabet. Aturan dalam pembentukan bahasa berfungsi untuk menentukan apakah kalimat tertentu legal

atau tidak legal. Sintak tidak peduli dengan makna atau isi dari suatu kalimat, semantiklah yang berfungsi untuk memaknainya.

Dalam bahasa python, terdapat berbagai macam sintak. Keyword merupakan salah satu dari bagian sintak. Keyword di dalam bahasa python merupakan reserved word yang berarti seluruh kata yang termasuk ke dalam keyword tidak dapat dijadikan sebagai nama dari suatu variabel, fungsi, class, dan berbagai identifier lainnya.

Berikut merupakan tabel keyword yang terdapat dalam bahasa pemrograman python.

Keywords in Python programming language				
False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Mengingat bahwa tidak semua keyword akan diperiksa, berikut disajikan tabel untuk beberapa keyword beserta karakteristiknya yang akan diimplementasikan dalam tugas besar ini.

NO.	Keyword	Spesifikasi/Karakteristik
1.	True, False, and, or, not	Dapat ditulis langsung sebagai pernyataan True atau False, dapat juga merupakan hasil dari operator perbandingan dari dua operan seperti ==, >, >=, <, <=, and, or. Dengan satu operan yaitu not. Dapat juga berupa hasil aritmatika dari True dan False seperti True + True yang tetap menghasilkan True
2.	None	Konstata spesial yang menyatakan nilai NULL. Seringkali diassign ke dalam variabel seperti var = None
3.	class	Dalam membuat class perlu di penulisan sebagai berikut: class <nama_class>:
4.	continue, break	Digunakan di dalam for dan while
5.	def	Mirip seperti class tetapi terdapat parameter, penulisannya sebagai berikut: def function_nama(parameters): Jumlah parameter dapat lebih dari satu.
6.	if, elif, else	Penulisan sebagai berikut if <condition>: <condition> dapat berada di dalam tanda kurung. if dapat berdiri sendiri, elif hanya dapat ditulis ketika if

		telah dinyatakan sebelumnya dan dapat ditulis berulang kali, sedangkan else berada di paling bawah setelah if atau elif dituliskan. elif diikuti dengan <condition>: sedangkan else hanya diikuti :
7.	in	pengecekan apakah suatu nilai berada di dalam list, tuple, atau string, contoh 5 in a, for i in "hello"
8.	while	penulisan mirip dengan if yaitu while <condition>:
9.	return	Digunakan di dalam fungsi. Penulisaannya return <nilai dengan type data apa pun>, dapat juga dengan mengembalikan banyak nilai, dengan cara memberi tanda koma di antara dua nilai kembalian
10.	with	Penulisan adalah with <statement>() as <var>:
11.	raise	Penulisan sebagai berikut: raise <nama_error>("<pesan error>")
12.	import, as, from	Sering kali digunakan bersamaan sebagai import <nama_module> as <nama_alias> from <spesifik_atribut>
13.	is	mengecek apakah dua variabel merupakan objek yang sama atau tidak
14.	pass	Merupakan null statement di dalam python, biasanya berada dialam fungsi atau class
15.	for	Secara general digunakan untuk melakukan looping. Cara penulisan dengan in sebagai berikut: for <nama_var> in <nama_list>:

Penamaan variabel juga merupakan salah satu dari sintak. Berikut merupakan aturan penamaan variabel pada bahasa pemrograman python

1. Nama variabel harus dimulai dengan huruf atau karakter underscore (_).
2. Nama variabel tidak dapat dimulai dengan angka
3. Nama variabel hanya boleh mengandung karakter alpanumerik dan underscore (A-z, 0-9, dan _)
4. Nama variabel adalah case-sensitive, maksudnya adalah variabel 'umur', 'Umur', dan 'UMUR' adalah variabel yang berbeda.

BAB 2

Hasil FA dan CFG

2.1. Pembuatan Finite Automata

Dalam membuat finite automata, kita perlu untuk memperhatikan aturan sintak pada penamaan sebuah variabel pada bahasa pemrograman python. Finite automata yang kami buat merupakan Deterministic-Finite Automata yang dapat dinyatakan sebagai

$$A = (\{q_0, q_1, q_2\}, \Sigma, \delta, q_0, q_1)$$

dengan $\Sigma = \{alfabets, numbers, asciiChars\}$

```
alfabets : ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a',
'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '_']
```

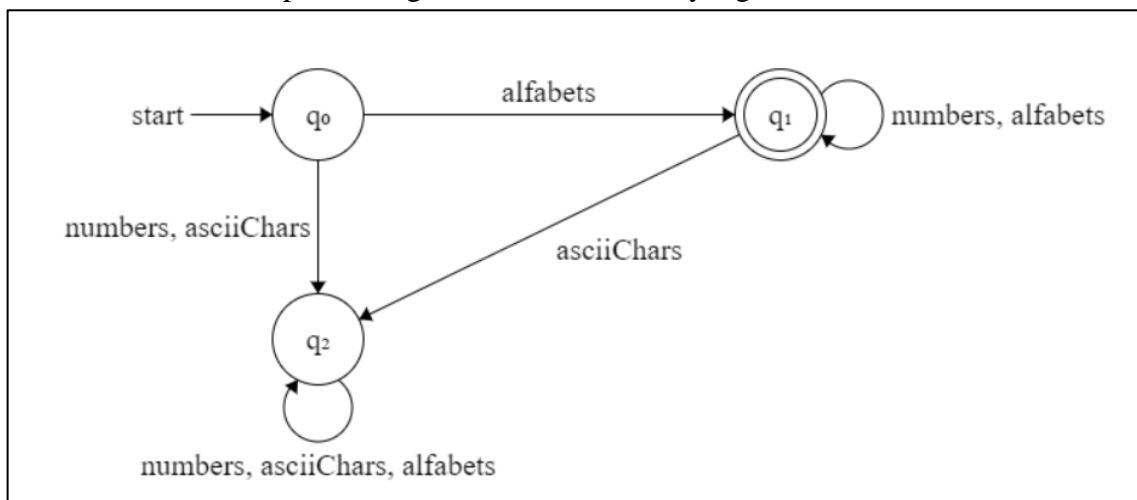
```
numbers : ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
asciiChars : ['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',',
'-', '.', '/', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^',
'_', '{', '|', '}', '~']
```

dan dengan δ sebagai berikut

δ	alfabets	numbers	asciiChars
$\rightarrow q_0$	q_1	q_2	q_2
$*$ q_1	q_1	q_1	q_2
q_2	q_2	q_2	q_2

Berikut merupakan diagram states dari DFA yang telah dibuat:



2.2. Pembuatan Context-Free Grammar

Context-Free Grammar kami buat secara manual seperti berikut.

$$G = (V, T, P, S)$$

Non-Terminal/Variabel Symbol:

S	MULTIVAR	VAR	VAL	BOOLEAN
OPS	STRING	STRING2	IF	ELIF
ELSE	PRINT	CONDITIONAL	RELATION	INPUT
BRACKET	WHILE	FOR	RANGE	DEF
CLASS	IMPORT	FROM	RETURN	RAISE
WITH	BREAK	PASS	CONTINUE	COMMENT
WITHVAR	METHOD			

Terminal Symbol:

+	-	*	/	%	and	or	variable
number	is	!	not	=	>	<	(
)	#	True	False	string	'	"	if
elif	else	:	print	while	for	in	range
def	class	import	as	from	return	raise	range
,	break	pass	continue	with	open	.	read
write							

Productions:

Productions	Result
S	S S VAR = MULTIVAR VAR + = MULTIVAR VAR - = MULTIVAR VAR * = MULTIVAR VAR / = MULTIVAR IF PRINT WHILE FOR DEF CLASS IMPORT FROM COMMENT WITH METHOD RETURN BREAK PASS CONTINUE RAISE;
MULTIVAR	VAR VAL MULTIVAR , MULTIVAR VAR OPS MULTIVAR MULTIVAR OPS VAR VAR OPS VAL;
VAR	variable;
VAL	number MULTIVAR OPS MULTIVAR MULTIVAR * * MULTIVAR MULTIVAR // MULTIVAR (MULTIVAR) BOOLEAN STRING;
RELATION	> < = != <= >=;
BOOLEAN	True False BOOLEAN and BOOLEAN BOOLEAN or BOOLEAN not BOOLEAN MULTIVAR is MULTIVAR MULTIVAR RELATION MULTIVAR BOOLEAN OPS BOOLEAN;
OPS	+ - * / %;
DEF	def VAR BRACKET : S DEF RETURN;
RETURN	return BOOLEAN return VAL;
CLASS	class VAR : S;
IMPORT	import VAR as VAR import VAR;
FROM	from VAR IMPORT;
STRING	" STRING2 " ' STRING2 ' STRING + STRING;
STRING2	string . is number and or not STRING2 STRING2;
COMMENT	" " " STRING2 " " " ' ' ' STRING2 ' ' ' # STRING2;
BRACKET	(MULTIVAR) ();
PRINT	print BRACKET;

IF	if CONDITIONAL IF ELIF IF ELSE IF RAISE IF BREAK IF PASS IF CONTINUE;
CONDITIONAL	(BOOLEAN) : S BOOLEAN : S;
RAISE	raise VAR BRACKET;
BREAK	break;
PASS	pass;
CONTINUE	continue;
ELIF	elif CONDITIONAL ELIF ELIF ELIF ELSE ELIF RAISE ELIF BREAK ELIF PASS ELIF CONTINUE;
ELSE	else : S ELSE RAISE ELSE BREAK ELSE PASS ELSE CONTINUE;
RANGE	range BRACKET;
FOR	for VAR in STRING : S for VAR in RANGE : S;
WITH	with open (STRING , STRING) as VAR : VAR = WITHVAR with open (STRING , STRING) as VAR : VAR = WITHVAR;
WITHVAR	VAR . read BRACKET VAR . write BRACKET;
METHOD	VAR = VAR . VAR BRACKET VAR = VAR BRACKET;
WHILE	while CONDITIONAL

2.3. Hasil Konversi CFG ke dalam CNF

VARIABLE	RESULT
S	S S VAR A1 VAR B1 VAR C1 VAR D1 VAR E1 J2 CONDITIONAL IF ELIF IF ELSE IF RAISE IF BREAK IF PASS IF CONTINUE K2 BRACKET Y1 CONDITIONAL E2 G11 E2 H11 S2 R1 DEF RETURN Q2 S1 P2 T1 P2 VAR N2 U1 M2 Z1 L2 A11 C2 I11 C2 J11 VAR M11 VAR N11 R2 BOOLEAN R2 VAL break pass continue I2 E11
Z2	= M2 Z3
A1	Z2 MULTIVAR
B1	OPS B2
B2	Z2 MULTIVAR open
C1	OPS C2
C2	Z2 MULTIVAR with
D1	OPS D2
D2	Z2 MULTIVAR in
E1	OPS E2
E2	Z2 MULTIVAR for
MULTIVAR	VAR MULTIVAR F1 VAR G1 MULTIVAR H1 VAR I1 number MULTIVAR J1 MULTIVAR K1 X2 L1 True False BOOLEAN M1 BOOLEAN N1 STRING2 BOOLEAN MULTIVAR O1 MULTIVAR P1 BOOLEAN Q1 M2 W1 L2 X1 STRING Y1
Y2	,
F1	Y2 MULTIVAR
G1	OPS MULTIVAR
H1	OPS VAR

I1	OPS VAL
VAR	variable
VAL	number MULTIVAR J1 MULTIVAR K1 X2 L1 True False BOOLEAN M1 BOOLEAN N1 STRING2 BOOLEAN MULTIVAR O1 MULTIVAR P1 BOOLEAN Q1 M2 W1 L2 X1 STRING Y1
J1	OPS J2
J2	OPS MULTIVAR if
K1	OPS K2
K2	OPS MULTIVAR print
X2	(
W2)
L1	MULTIVAR W2
RELATION	> < Z2 Z2 U2 Z2 RELATION Z2 RELATION Z2
U2	!
BOOLEAN	True False BOOLEAN M1 BOOLEAN N1 STRING2 BOOLEAN MULTIVAR O1 MULTIVAR P1 BOOLEAN Q1
M1	STRING2 BOOLEAN
N1	STRING2 BOOLEAN
O1	STRING2 MULTIVAR
P1	RELATION MULTIVAR
Q1	OPS BOOLEAN
OPS	+ - * / %
T2	: O2 VAR
S2	def T2 S
DEF	S2 R1 DEF RETURN
R1	VAR R2
R2	BRACKET R3 return
R3	T2 S
RETURN	R2 BOOLEAN R2 VAL
Q2	class
CLASS	Q2 S1
S1	VAR S2
P2	import
O2	as
IMPORT	P2 T1 P2 VAR
T1	VAR T2
N2	from
FROM	N2 U1
U1	VAR IMPORT
M2	"
STRING	M2 W1 L2 X1 STRING Y1
W1	STRING2 M2
L2	'
X1	STRING2 L2
Y1	OPS STRING while
STRING2	string . is number and or not STRING2 STRING2 CONTENT
COMMENT	M2 Z1 L2 A11

Z1	M2 Z2 write
Z3	STRING2 Z4
Z4	M2 Z5
Z5	M2 M2
A11	L2 A12
A12	L2 A13
A13	STRING2 A14
A14	L2 A15
A15	L2 L2
BRACKET	X2 B11 X2 W2
B11	MULTIVAR W2
PRINT	K2 BRACKET
IF	J2 CONDITIONAL IF ELIF IF ELSE IF RAISE IF BREAK IF PASS IF CONTINUE
CONDITIONAL	X2 C11 BOOLEAN D11
C11	BOOLEAN C12
C12	W2 C13
C13	T2 S
D11	T2 S
I2	raise
RAISE	I2 E11
E11	VAR BRACKET
BREAK	break
PASS	pass
CONTINUE	continue
H2	elif
ELIF	H2 CONDITIONAL ELIF ELIF ELIF ELSE ELIF RAISE ELIF BREAK ELIF PASS ELIF CONTINUE
G2	else
ELSE	G2 F11 ELSE RAISE ELSE BREAK ELSE PASS ELSE CONTINUE
F11	T2 S
F2	range
RANGE	F2 BRACKET
FOR	E2 G11 E2 H11
G11	VAR G12
G12	D2 G13
G13	STRING G14
G14	T2 S
H11	VAR H12
H12	D2 H13
H13	RANGE H14
H14	T2 S
WITH	C2 I11 C2 J11
I11	B2 I12
I12	X2 I13

I13	STRING I14
I14	Y2 I15
I15	STRING I16
I16	W2 I17
I17	O2 I18
I18	VAR I19
I19	T2 I110
I110	VAR I111
I111	Z2 WITHVAR
J11	B2 J12
J12	X2 J13
J13	STRING J14
J14	Y2 J15
J15	STRING J16
J16	W2 J17
J17	O2 J18
J18	VAR J19
J19	T2 J110
J110	VAR J111
J111	Z2 WITHVAR
A2	read
WITHVAR	VAR K11 VAR L11
K11	STRING2 K12
K12	A2 BRACKET
L11	STRING2 L12
L12	Z1 BRACKET
METHOD	VAR M11 VAR N11
M11	Z2 M12
M12	VAR M13
M13	STRING2 M14
M14	VAR BRACKET
N11	Z2 N12
N12	VAR BRACKET
WHILE	Y1 CONDITIONAL
S0	S S VAR A1 VAR B1 VAR C1 VAR D1 VAR E1 J2 CONDITIONAL IF ELIF IF ELSE IF RAISE IF BREAK IF PASS IF CONTINUE K2 BRACKET Y1 CONDITIONAL E2 G11 E2 H11 S2 R1 DEF RETURN Q2 S1 P2 T1 P2 VAR N2 U1 M2 Z1 L2 A11 C2 I11 C2 J11 VAR M11 VAR N11 R2 BOOLEAN R2 VAL break pass continue I2 E11

BAB 3

Implementasi dan Pengujian

3.1. Spesifikasi Program

3.1.1. File CFG2CNF.py

File CFG2CNF.py berfungsi untuk mengubah suatu txt cfg.txt yang berisi cfg bahasa python menjadi bentuk cnf kemudian di save di file cnf.txt. Untuk algoritma ini kami mengambil referensi dari <https://github.com/adelmassimo/CFG2CNF>. Fungsi-fungsi dan prosedur yang kami gunakan adalah sebagai berikut:

No.	Fungsi / Prosedur	Tujuan fungsi/prosedur
1.	createDict	Membuat kamus yang memuat semua terminal dari CFG
2.	isUnitary	Fungsi boolean yang menentukan jika suatu aturan produksi memiliki tepat satu buah symbol terminal dan non-terminal pada daftar variable tertentu.
3.	isSimple	Kegunaan sama seperti isUnitary tetapi hanya diperlukan input aturan produksi saja dan melihat daftar variable yang sudah tersimpan.
4.	START	Membuat aturan produksi $S_0 \rightarrow S$
5.	TERM	Memisahkan aturan produksi yang menghasilkan variabel dan terminal bersebelahan, kemudian dipisah menjadi 2 aturan. (Contoh: $A \rightarrow Bb$ diubah menjadi $A \rightarrow BX, X \rightarrow b$)
6.	BIN	Menghapus aturan produksi non-Unitary pada suatu kumpulan aturan produksi
7.	FindOutlaws	Mencari kemudian mengeliminasi variabel useless.
8.	rewrite	Menulis ulang file CFG menjadi format yang dapat dibaca satu-persatu seperti pada suatu array.
9.	DEL	Menghapus aturan produksi non-terminal yang masih ada
10.	unit_routine	Memeriksa jika bentuk suatu aturan produksi unary atau single
11.	UNIT	Menghapus aturan produksi unit yang masih ada
12.	convertToMap	Melakukan konversi kumpulan aturan produksi menjadi berbentuk map
13.	cleanAlphabet	Membersihkan list yang berisi terminal dan variabel agar dapat diproses
14.	cleanProduction	Membersihkan list yang berisi aturan produksi agar dapat diproses

15.	loadModel	Membersihkan list dari suatu CFG dengan menggunakan fungsi cleanAlphabet dan cleanProduction
16.	writeFormat	Menulis ulang CNF yang sudah diproduksi menjadi format yang lebih rapi

3.1.2. File main.py

File ini merupakan file program utama yang akan dijalankan oleh pengguna.

No.	Proses	Keterangan
1.	baca file python	Pengguna dapat menuliskan pada terminal sebagai py main.py <inputFile.py>, apabila hanya py main.py, program akan membaca file default yaitu tesInput.py
2.	tokenized code	menghilangkan inline komentar (#) dan akan mengubah file menjadi token token yang siap diproses
3.	CYK	Pemrosesan token untuk diperiksa sintak dari program
4.	checking variable name using DFA	pengambilan variabel dari file input lalu diproses melalui DFA untuk diperiksa kevalidannya

3.1.3. File cyk.py

File ini berfungsi untuk melakukan algoritma CYK dan memeriksa apakah file input yang sudah ditokenisasi dapat dibentuk dari CNF yang sudah didefinisikan sebelumnya.

NO.	Fungsi/Prosedur	Keterangan
1.	LoadCNF	Input prosedur ini adalah file .txt dari CNF yang dihasilkan oleh CFG. Program akan memisahkan rules CNF dan memasukkannya ke dalam dictionary, jika key grammar belum ada, maka akan dimasukkan key baru dan valuenya, jika sudah ada, maka akan dimasukkan value ke key yang sudah ada (di-append).
2.	cyk	Input fungsi ini adalah hasil tokenisasi file input. Program akan menginisialisasi table CYK dengan value pada grammar menggunakan key yang bersesuaian dengan input token, jika key tidak ditemukan, maka akan dicari pattern menggunakan <i>regular expression</i> dan jika sesuai, table CYK akan diinisialisasi dengan value grammar menggunakan key tersebut. Setelah table diinisialisasi, akan dilakukan iterasi untuk mencari kombinasi grammar, jika key ditemukan pada grammar, table CYK akan ditambahkan dengan valuenya. Program akan menghasilkan output True jika input token dapat dimulai dari startsymbol dan False jika tidak.

3.1.4. File tokenizationVar.py

File ini berfungsi untuk melakukan tokenisasi khusus pada nama variabel.

NO.	Fungsi/Prosedur	Keterangan
1.	tokenizedVar	Input adalah string hasil baca file dan akan mengembalikan list berisi nama variabel. Dalam program ini, variabel yang diambil adalah assign variable seperti pada umumnya, nama class dan nama fungsi.

3.1.5. File variableChecking.py

File ini berfungsi untuk melakukan pemeriksaan nama variabel menggunakan finite automata

NO.	Fungsi/Prosedur	Keterangan
1.	tokenDFACheck	input adalah token yang berupa string. Prosedur ini akan memeriksa satu persatu karakter dari token sebagai input pada finite automata
2.	checkingNamingVariable	input adalah list dari variabel. Akan diperiksa satu persatu variabel, apabila ada penamaan yang tidak valid, program akan memberitahu variabel tersebut

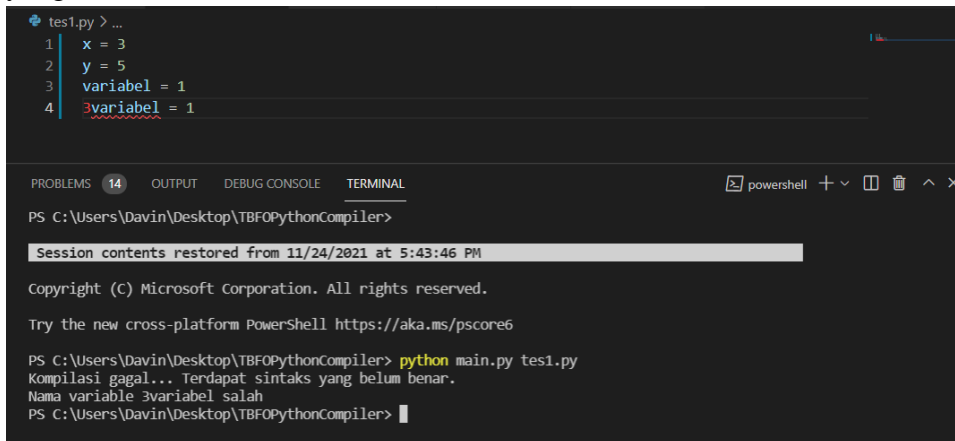
3.2. Hasil Pengujian

Berikut pengujian file inputAcc.py yang ada di Spek Tubes, program kami menampilkan pesan bahwa sintaks file diterima dan diproduksi CFG. Pada percobaan ini grammar produksi yang dipakai adalah DEF, VAR, VAL, MULTIVAR, BRACKET, COMMENT, STRING, IF, ELIF, ELSE, CONDITIONAL, BOOLEAN, RETURN, RELATION, dan OPERATION.

Dapat dilihat pada akar dari table CYK mengandung startsymbol "S" karena terdapat kombinasi grammar yang dapat mencapai startsymbol, dengan demikian sintaks file diterima.

Dapat dilihat pada akar dari table CYK kosong dan tidak mengandung startsymbol “S” karena terdapat kombinasi grammar yang tidak dapat mencapai startsymbol, dengan menggunakan FA dapat dianalisis bahwa kesalahan sintaks terdapat pada penamaan variabel.

Berikut contoh pengujian file yang salah karena inisialisasi nama variabel yang salah:

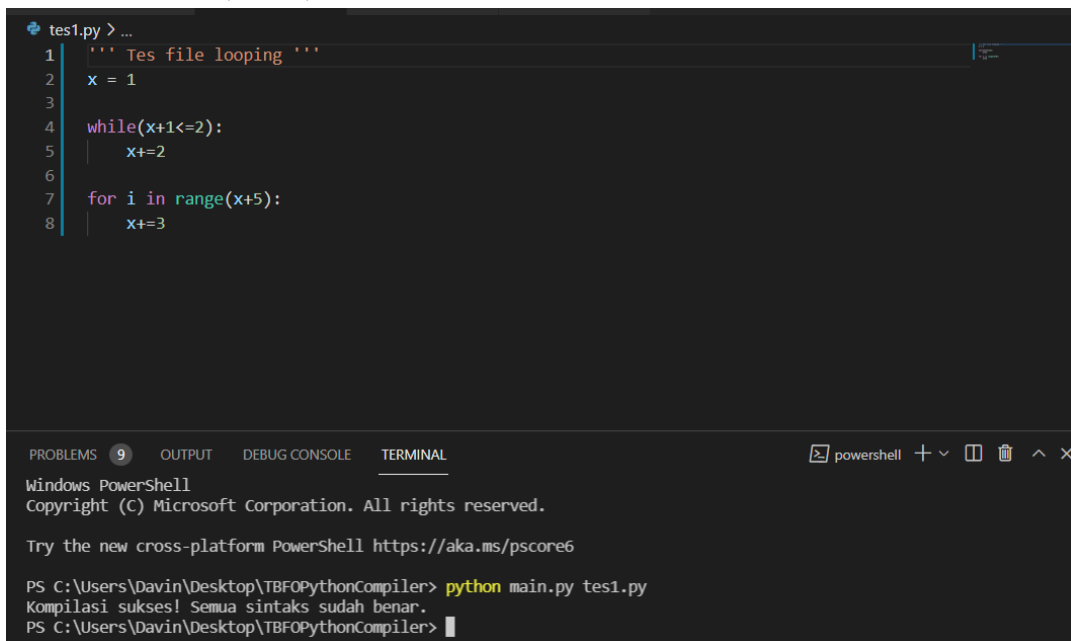


```
tes1.py > ...
1 x = 3
2 y = 5
3 variabel = 1
4 3variabel = 1

PROBLEMS 14 OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Davin\Desktop\TBFOPythonCompiler>
Session contents restored from 11/24/2021 at 5:43:46 PM
Copyright (c) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Davin\Desktop\TBFOPythonCompiler> python main.py tes1.py
Kompilasi gagal... Terdapat sintaks yang belum benar.
Nama variable 3variabel salah
PS C:\Users\Davin\Desktop\TBFOPythonCompiler>
```

Program mengirimkan output variabel salah karena terdapat angka yang berada pada awalan variabel. Pada pengecekan DFA, nama variabel yang seperti ini akan berakhir pada state 2, yaitu state buangan sehingga tidak akan pernah mencapai final state.

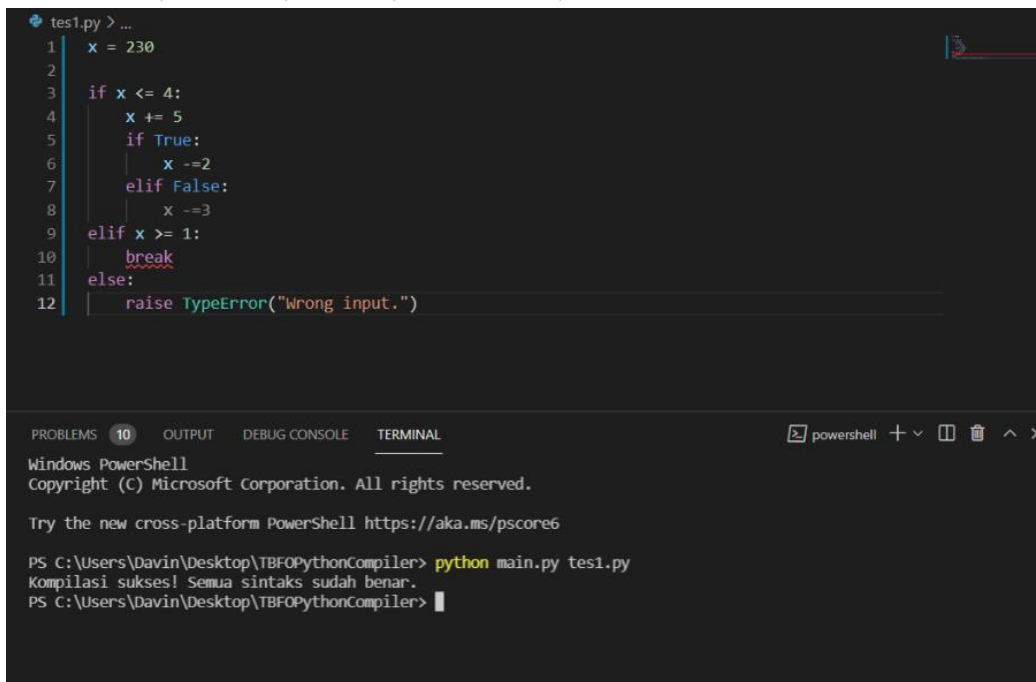
Berikut pengujian file untuk kode looping, program kami menampilkan pesan bahwa sintaks file diterima dan diproduksi CFG. Pada percobaan ini grammar yang dipakai adalah produksi VAR, VAL, MULTIVAR, OPS, WHILE, BRACKET, CONDITIONAL, FOR, dan RANGE.



```
tes1.py > ...
1 ''' Tes file looping '''
2 x = 1
3
4 while(x+1<=2):
5     x+=2
6
7 for i in range(x+5):
8     x+=3

PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Davin\Desktop\TBFOPythonCompiler> python main.py tes1.py
Kompilasi sukses! Semua sintaks sudah benar.
PS C:\Users\Davin\Desktop\TBFOPythonCompiler>
```

Berikut pengujian file untuk kode conditional, program kami menampilkan pesan bahwa sintaks file diterima dan diproduksi CFG. Pada percobaan ini grammar yang dipakai adalah produksi VAR, VAL, MULTIVAR, IF, ELIF, CONDITIONAL, BOOLEAN, BREAK, RAISE, BRACKET, dan STRING.



```

tes1.py > ...
1  x = 230
2
3  if x <= 4:
4      x += 5
5      if True:
6          x -= 2
7      elif False:
8          x -= 3
9  elif x >= 1:
10     break
11 else:
12     raise TypeError("Wrong input.")

```

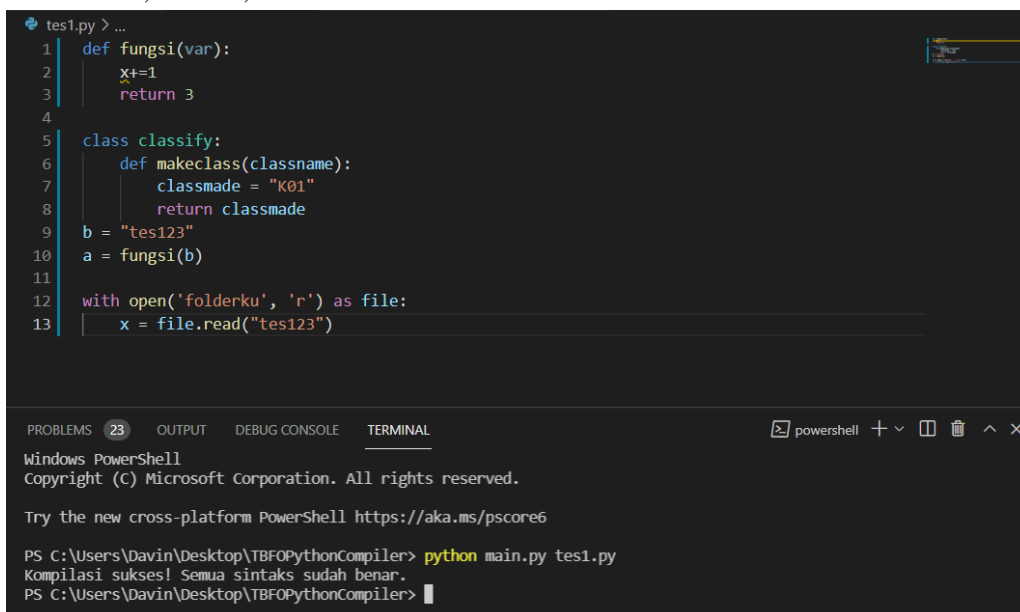
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\Davin\Desktop\TBFOPythonCompiler> python main.py tes1.py
Kompilasi sukses! Semua sintaks sudah benar.
PS C:\Users\Davin\Desktop\TBFOPythonCompiler>

Berikut pengujian file untuk kode definisi fungsi / prosedur, kelas, dan fungsi with, program kami menampilkan pesan bahwa sintaks file diterima dan diproduksi CFG. Pada pengujian ini kami menggunakan grammar produksi VAR, VAL, MULTIVAR, DEF, BRACKET, OPS, RETURN, CLASS, STRING, RETURN, METHOD, WITH, dan WITHVAR.



```

tes1.py > ...
1  def fungsi(var):
2      x+=1
3      return 3
4
5  class classify:
6      def makeclass(classname):
7          classmade = "K01"
8          return classmade
9  b = "tes123"
10 a = fungsi(b)
11
12 with open('folderku', 'r') as file:
13     x = file.read("tes123")

```

PROBLEMS 23 OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\Davin\Desktop\TBFOPythonCompiler> python main.py tes1.py
Kompilasi sukses! Semua sintaks sudah benar.
PS C:\Users\Davin\Desktop\TBFOPythonCompiler>

BAB 4

Penutup

4.1. Kesimpulan

1. Context-Free Grammar yang telah diubah ke dalam bentuk Chomsky Normal Form dapat digunakan sebagai pemeriksa sintak python melalui algoritma Cocke-Younger-Kasami
2. Penggunaan finite automata dapat digunakan sebagai pemeriksaan nama variabel pada bahasa pemrograman python
3. Program yang kami buat telah dapat melakukan pemeriksaan sintak python dalam skala kecil, meskipun tidak mencakupi keseluruhan sintak dan masih terdapat beberapa bug di dalamnya.
4. Pada pemeriksaan nama variabel, program kami dapat berjalan dengan baik hanya saja masih terbatas pada assign variable, nama class, dan nama fungsi.

4.2. Saran

1. Meningkatkan kesabaran, ketelitian, dan waktu dalam pembuatan CFG sehingga dapat mencakup keseluruhan sintak pada pemrograman python
2. Meningkatkan proses tokenisasi nama variabel sehingga dapat mencakup lebih banyak tempat variabel berada, contohnya nama parameter, argumen, variabel di dalam condition if dan while, serta variabel di dalam for.

4.3. Link Repository Github

<https://github.com/fdavin/TBFOPythonCompiler>

4.4. Pembagian Tugas

NIM	Nama	Pembagian Tugas
13520025	Fransiskus Davin Anwari	<ol style="list-style-type: none">1. Membuat file CFG.txt2. Mengubah CFG menjadi bentuk CNF3. Laporan 1.2, 1.3, 2.2, 2.3
13520031	Taufan Fajarama Putrawansyah R	<ol style="list-style-type: none">1. Menerapkan algoritma CYK2. Laporan 1.4, 3.1, 3.2
13520033	Tri Sulton Adila	<ol style="list-style-type: none">1. Membaca file input python2. Tokenisasi file untuk pemeriksaan sintak3. Tokenisasi file untuk pemeriksaan nama variabel4. Membuat DFA dan algoritma penanganan nama variabel5. Laporan 1.1, 1.5, 2.1, 3.1, 4.1, 4.2

DAFTAR REFERENSI

- A guide to parsing: Algorithms and terminology*. Strumenta. (2021, February 2). Retrieved November 24, 2021, from <https://tomassetti.me/guide-parsing-algorithms-terminology/>.
- Adelmassimo. (n.d.). *Adelmassimo/CFG2CNF: Python tool able to convert a context free grammar in Chomsky normal form*. GitHub. Retrieved November 22, 2021, from <https://github.com/adelmassimo/CFG2CNF>.
- ASCII table*. ASCII Table - ASCII Character Codes, HTML, Octal, Hex, Decimal. (n.d.). Retrieved November 24, 2021, from <https://www.asciitable.com/>.
- Aulia, A. (n.d.). *Konversi CFG Menjadi CNF*. School of Computer Science. Retrieved November 23, 2021, from <https://socs.binus.ac.id/2018/12/20/konversi-cfg-menjadi-cnf/>.
- Kang, C. (2020, July 15). *Regular expressions and word tokenization*. Chan's Jupyter. Retrieved November 24, 2021, from https://goodboychan.github.io/python/datacamp/natural_language_processing/2020/07/15/01-Regular-expressions-and-word-tokenization.html.
- Library.binus.ac.id*. (2011). Retrieved November 24, 2021, from <http://library.binus.ac.id/eColls/eThesisdoc/Bab2/2011-1-00604-mtif%202.pdf>.
- List of keywords in Python*. Programiz. (n.d.). Retrieved November 24, 2021, from <https://www.programiz.com/python-programming/keyword-list>.
- Luthfi, I. (n.d.). *Aplikasi Program Dinamis dalam Algoritma Cocke-Younger-Kasami (CYK)*. Retrieved November 24, 2021, from <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2007-2008/Makalah2008/MakalahIF2251-2008-079.pdf>.
- Python Variable Names*. W3Schools. (n.d.). Retrieved November 24, 2021, from https://www.w3schools.com/python/gloss_python_variable_names.asp.
- Said, F. E. (2011, June 15). *Context free grammar (CFG)*. Fairuz el Said. Retrieved November 23, 2021, from <https://fairuzelsaid.wordpress.com/2011/06/16/tbo-context-free-grammar-cfg/>.
- Sivakumar, B. (2020, October 6). *Extracting words from a string in python using regex*. Medium. Retrieved November 24, 2021, from <https://medium.com/quantrium-tech/extracting-words-from-a-string-in-python-using-regex-dac4b385c1b8>.