Laporan Tugas Besar IF 2121 Logika Komputasional

***Compiler* Bahasa Python**

Disusun Oleh :
Dolphin Terbang

1. William Manuel Kurniawan	**13520020**
2. Fadil Fauzani	**13520032**
3. Ilham Pratama	**13520041**

Institut Teknologi Bandung
Bandung
2021

Daftar Isi

BAB I
Dasar Teori

1.1 Finite Automata (FA)

Finite Automata (FA) adalah model matematika yang dapat menerima input dan mengeluarkan output yang memiliki state yang berhingga banyaknya dan dapat berpindah dari satu state ke state lainnya berdasarkan input dan fungsi transisi. Finite state automata tidak memiliki tempat penyimpanan/memory, hanya bisa mengingat state terkini.
Finite State Automata dinyatakan oleh pasangan 5 tuple, yaitu:

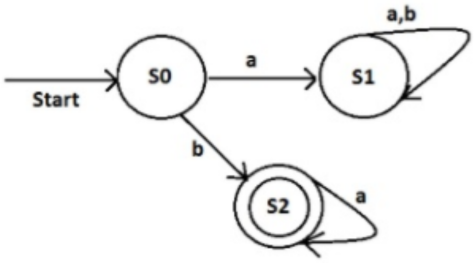
$$M=(Q, \Sigma, \delta, S, F)$$

*keterangan :
Q = himpunan state
Σ = himpunan simbol input
δ = fungsi transisi δ : Q × Σ
S = state awal / initial state , S ∈ Q
F = state akhir, F ⊆ Q

Terdapat 2 jenis FA yaitu:

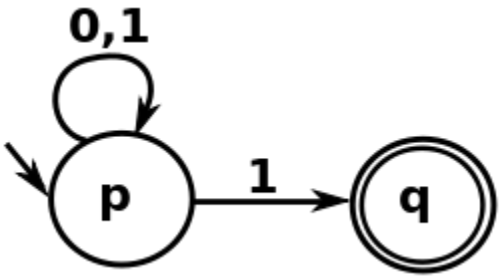
1. *Deterministic Finite Automata* (DFA)

   DFA artinya dari suatu state ada tepat satu state berikutnya untuk setiap simbol masukan yang diterima. Deterministik artinya tertentu/sudah tertentu fungsi transisinya. Jadi tidak ada suatu simbol yang sama dalam suatu state yang mengarah kedua atau lebih state. Contoh DFA adalah :



2. *Nondeterministic Finite Automata* (NFA)

NFA adalah kebalikan dari DFA. Jika di DFA tidak boleh ada 1 simbol yang mengarah ke dua atau lebih state pada suatu state, maka di NFA hal tersebut boleh. Di NFA boleh ada satu simbol yang mengarah ke 2 / lebih state.



1.2 Context Free Grammar (CFG)

Context Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa dan CFG juga bisa diartikan sebagai sebuah tata bahasa dimana tidak terdapat pembatasan pada hasil produksinya atau CFG juga bisa diartikan sebagai suatu. Contoh aturan produksi pada CFG adalah :

$$\alpha \rightarrow \beta$$

batasannya hanyalah ruas kiri (α) adalah sebuah simbol variabel. Sedangkan contoh aturan produksi CFG adalah sebagai berikut.

$$\mathbf{S \rightarrow aB}$$
$$\mathbf{B \rightarrow c}$$

CFG juga memiliki suatu bentuk umum. Bentuk umum dari CFG adalah

$$\mathbf{G = (V, T, P, S)}$$

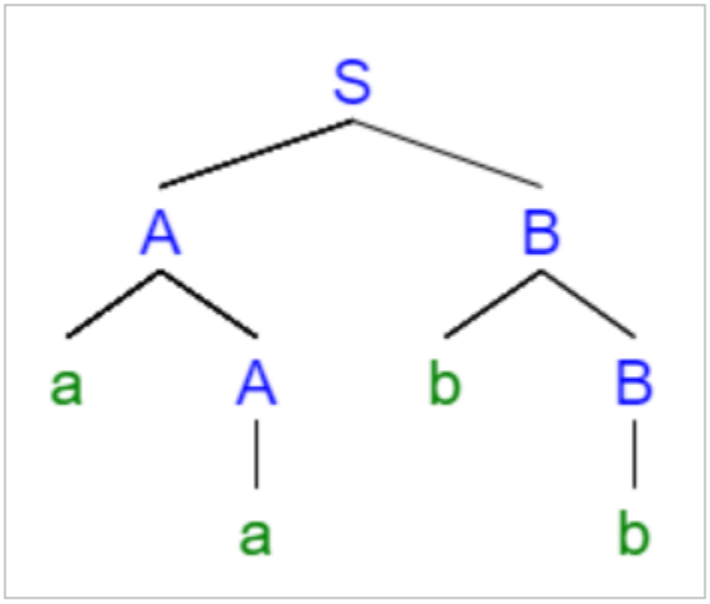
*keterangan :
G = CFG
V = himpunan variabel
T = terminal
P = himpunan produksi
S = start symbol

Dalam CFG ada suatu proses yang bernama proses parsing. Proses parsing adalah proses pembacaan string dalam bahasa sesuai CFG tertentu, proses ini harus mematuhi aturan produksi dalam CFG tersebut. Context Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan didefinisikan 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 diturunkan menjadi terminal sampai tidak ada yang belum tergantikan.

Contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

$$S \rightarrow AB$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bB \mid b$$

Maka jika ingin dicari gambar parse tree dengan penerimaan string “aabb”, dapat diturunkan sebagai berikut.



Proses penurunan / parsing bisa dilakukan dengan 2 cara, yaitu :

- *leftmost derivation* : simbol variabel paling kiri yang di perluas terlebih dahulu.
- *rightmost derivation* : simbol variabel paling kanan yang diperluas terlebih dahulu.

Misalnya, terdapat suatu CFG G=({A,B,S}, {a,b}, P, S}

Dengan aturan produksi P :

$S \rightarrow aAB$

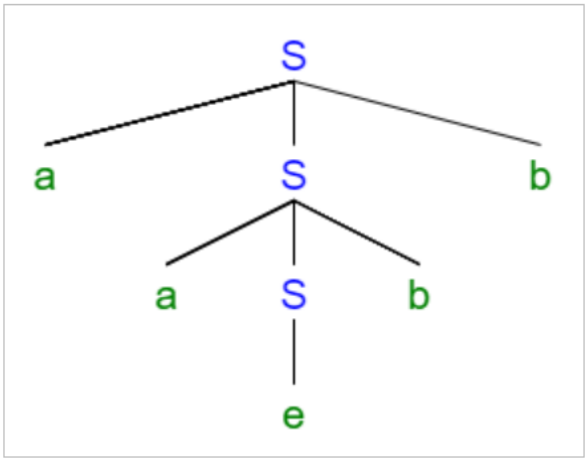
A → bBb
B → A | ε

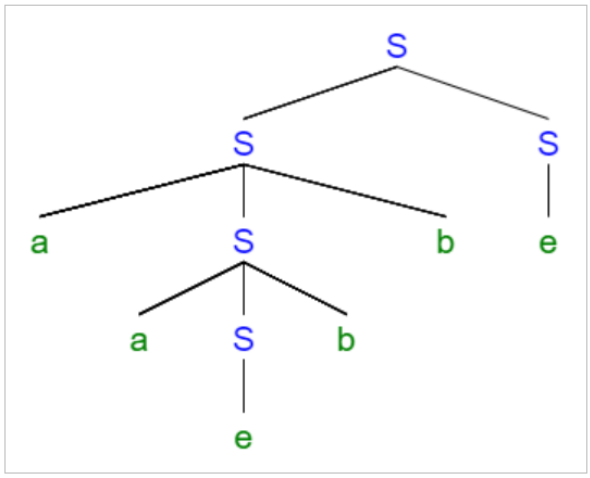
Untuk menghasilkan string “abbbb” dapat dilakukan dengan *leftmost derivation* atau dengan *rightmost derivation*.
leftmost derivation : S → aAB → abBbB → abAbB → abbBbbB → abbbbB → abbbb
rightmost derivation : S → aAB → aA → abBb → abAb → abbBbb → abbbb

Selain parsing, pada CFG juga dikenal istilah *ambiguitas*. *Ambiguitas* terjadi bila terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu string. Contohnya terdapat suatu tata bahasa sebagai berikut.

S → SS | aSb | ε

Untuk memperoleh string “aabb”, pohon penurunannya ada 2 yaitu





Karena ada 2 pohon penurunannya penerimaan string “aabb” disebut *ambiguitas*.

1.3 Chomsky Normal Form (CNF)

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 konteks tersebut:

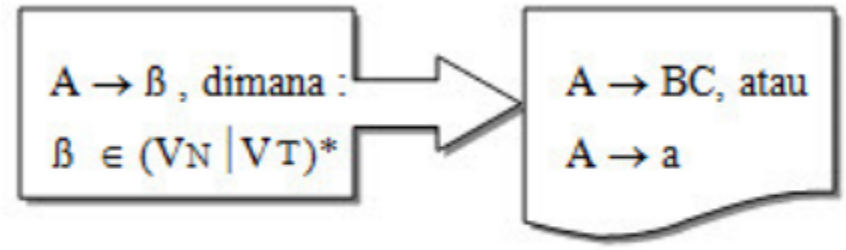
- Tidak memiliki produksi useless
- Tidak memiliki produksi unit
- Tidak memiliki produksi ε

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

A → BC atau A → a

Transformasi CFG ke CNF

Transformasi CFG ke CNF adalah transformasi sebagai berikut



Transformasi CFG Ke CNF

Aturan produksi dalam bentuk normal Chomsky ruas kanannya tepat berupa sebuah terminal atau dua variabel.

Misalkan:

- A → BC
- A → b

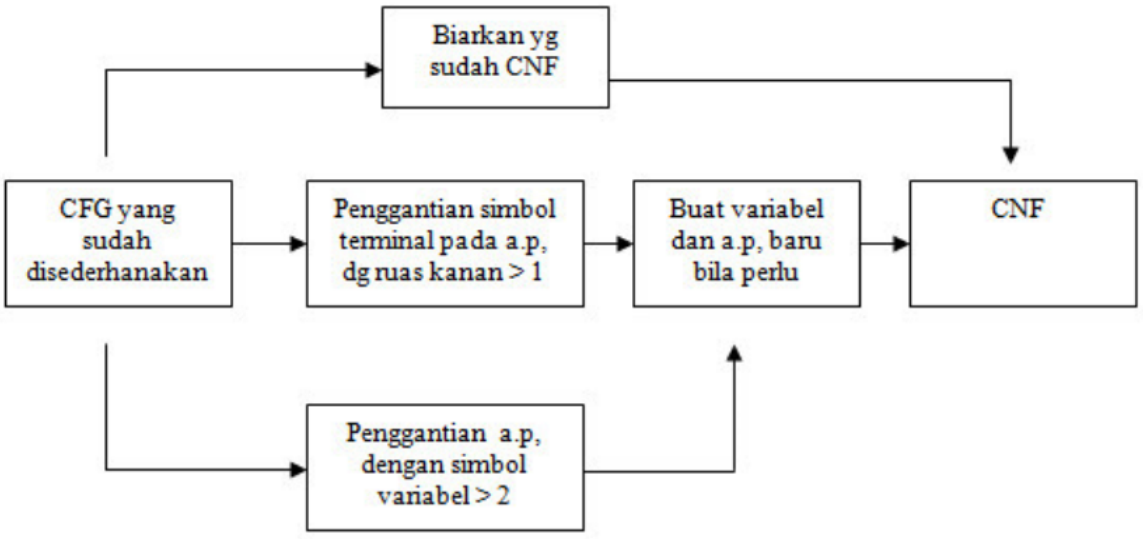
- B → a
- C → BA | d

Pembentukan CNF

Langkah-langkah pembentukan bentuk normal Chomsky secara umum sebagai berikut:

1. Biarkan aturan produksi yang sudah dalam bentuk normal Chomsky
2. Lakukan penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan panjang ruas kanan > 1
3. Lakukan penggantian aturan produksi yang ruas kanannya memuat > 2 simbol variabel
4. Penggantian-penggantian tersebut bisa dilakukan berkali-kali sampai akhirnya semua aturan produksi dalam bentuk normal Chomsky
5. Selama dilakukan penggantian, kemungkinan kita akan memperoleh aturan-aturan produksi baru, dan juga memunculkan simbol-simbol variabel baru

Bisa dilihat tahapan-tahapan tersebut pada gambar berikut:



Langkah-langkah pembentukan bentuk normal Chomsky

Konversi dari CFG ke CNF

Aturan produksi yang sudah dalam bentuk normal Chomsky:
$A \rightarrow a$
$B \rightarrow b$
Dilakukan penggantian aturan produksi yang belum bentuk CNF :

$S \rightarrow bA \Rightarrow S \rightarrow P_1A$
$S \rightarrow aB \Rightarrow S \rightarrow P_2B$
$A \rightarrow bAA \Rightarrow A \rightarrow P_1AA \Rightarrow A \rightarrow P_1P_3$
$A \rightarrow aS \Rightarrow A \rightarrow P_2S$
$B \rightarrow aBB \Rightarrow B \rightarrow P_2BB \Rightarrow B \rightarrow P_2P_4$
$B \rightarrow bS \Rightarrow B \rightarrow P_1S$

Terbentuk aturan produksi dan simbol variabel baru:

$P_1 \rightarrow b$
$P_2 \rightarrow a$
$P_3 \rightarrow AA$
$P_4 \rightarrow BB$

Hasil akhir aturan produksi dalam bentuk normal Chomsky :

$A \rightarrow a$
$B \rightarrow b$
$S \rightarrow P_1A$
$S \rightarrow P_2B$
$A \rightarrow P_1P_3$
$A \rightarrow P_2S$
$B \rightarrow P_2P_4$
$B \rightarrow P_1S$
$P_1 \rightarrow b$

$P_2 \rightarrow a$
$P_3 \rightarrow AA$
$P_4 \rightarrow BB$

$P_1 = P$, $P_2 = Q$, $P_3 = R$, $P_4 = T$

sehingga aturan produksinya menjadi:
$S \rightarrow PA$
$S \rightarrow QB$
$A \rightarrow PR$
$A \rightarrow QS$
$A \rightarrow a$
$B \rightarrow QT$
$B \rightarrow PS$
$B \rightarrow b$
$P \rightarrow b$
$Q \rightarrow a$
$R \rightarrow AA$
$T \rightarrow BB$

1.4 Cocke Younger Kasami (CYK)

Cocke Younger Kasami (CYK) adalah salah satu Algoritma parsing yang digunakan untuk pengujian membership suatu string pada suatu CFG yang sudah dalam bentuk CNF. Algoritma CYK ditemukan oleh 3 orang yang berbeda pada saat yang bersamaan, yaitu John Cocke, Daniel Younger, dan Tadao Kasami, dari ketiga nama tersebutlah Algoritma ini dinamakan. Algoritma CYK merupakan salah satu Algoritma yang cepat dengan kompleksitas $O(N^3)$, dengan N adalah panjang string yang dipakai dalam CYK.

Contoh penggunaan CYK

CNF:

$S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$

Dengan string yang diuji adalah “aabaa”, pertama kita buat tabel untuk membantu melakukan uji.

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

Didapatkan string tersebut tidak dapat diturunkan dari aturan produksi S, maka string “aabaa” tidak terdapat pada CNF.

Coba uji kembali dengan string “baaba”.

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

Didapatkan string tersebut dapat diturunkan dari aturan produksi S, maka string “aabaa” terdapat pada CNF.

BAB II
Analisis dan Dekomposisi Persoalan

2.1 Dekomposisi CFG

Berikut adalah CFG yang kami buat guna menjalankan *compiler* yang kamu buat.
Bentuk umum CFG :

G = (V, T, P, S)

V = variabel

START	VAR	VAL	BOOL	BOOLOP	OP
CALC	IF	ELIF	ELSE	WHILE	FOR
RANGE	COMMENT	DEF	CLASS	RETURN	FROM
IMPORT	CONTENT	WORD	START2	IF2	ELIF2
ELSE2	START3	IF3	ELIF3	ELSE3	PRINT
ARRAY	CONTENT2	VAR2	RAISE	FUNCTION	WITH

T = terminal

+	-	*	/	=	‘
(	)	[	]	,	:
!	<	>	and	as	break
class	continue	def	elif	else	for
from	if	import	in	is	not
or	pass	return	while	var	num
True	False	word	range	print	none
raise	with				

P = production

START	START START | VAR = CALC | VAR = VAL | VAR = NONE | VAR = BOOL | VAR = CONTENT | VAR = ARRAY | VAR = FUNCTION | IF | DEF | CLASS | FOR | WHILE | IMPORT | FROM | COMMENT | VAR + = VAL | VAR - = VAL | VAR * = VAL | VAR / = VAL | PRINT | RAISE | FUNCTION | WITH;
VAR	var;
VAL	num | WORD;
CONTENT	CONTENT CONTENT | VAL | VAR | CALC | (CONTENT) | WORD;
BOOL	True | False | CONTENT BOOLOP CONTENT | CONTENT in CONTENT | CONTENT is CONTENT | CONTENT or CONTENT | CONTENT and CONTENT | not CONTENT | BOOL or BOOL | BOOL and BOOL | not BOOL | (BOOL);
BOOLOP	== | != | <= | >= | < | >;
OP	+ | - | * | / | %;
CALC	VAL OP VAL | VAR OP VAL | VAL OP VAR | VAR OP VAR;
IF	if (BOOL) : START | if BOOL : START | IF ELIF | IF ELSE;
ELIF	elif (BOOL) : START | elif BOOL : START | ELIF ELIF | ELIF ELSE;
ELSE	else : START;
WHILE	while (BOOL) : START3 | while BOOL : START3 | while (BOOL) : break | while BOOL : break | while (BOOL) : continue | while BOOL : continue | while (BOOL) : pass | while BOOL : break | WHILE continue | WHILE pass | WHILE break;
FOR	for VAR in VAR : START3 | for VAR in RANGE | for VAR in VAR : break | for VAR in VAR : pass | for VAR in VAR : continue;
RANGE	range (num) : START3 | range (num , num) : START3 | range (num , num , num) : START3 | range (num) : break | range (num , num) : break | range (num , num , num) : break | range (num) : continue | range (num , num) : continue | range (num , num , num) : continue | range (num) : pass | range (num , num) : pass | range (num , num , num) : pass | RANGE continue | RANGE pass | RANGE break;
COMMENT	' ' ' word ' ' ' | " " " word " " ";
DEF	def VAR (CONTENT) : START2 | def VAR ( ) : START2 | DEF RETURN |

	def VAR (CONTENT) : RETURN | def VAR ( ) : RETURN | def VAR (CONTENT) : pass | def VAR ( ) : pass | DEF pass;
CLASS	class VAR : START | class VAR : pass;
RETURN	return BOOL | return VAL | return WORD;
FROM	from VAR IMPORT;
IMPORT	import VAR2 | import VAR as VAR;
VAR2	VAR | VAR2 , VAR2;
WORD	" word " | ' word ' | WORD + WORD | " " | ' ';
START2	START2 START2 | VAR = CALC | VAR = VAL | VAR = ARRAY | VAR = NONE | VAR = FUNCTION | IF2 | FOR | WHILE | COMMENT | VAR + = VAL | VAR - = VAL | VAR * = VAL | VAR / = VAL | PRINT | RAISE | FUNCTION | WITH;
IF2	if (BOOL) : START2 | if BOOL : START2 | IF2 ELIF2 | IF2 ELSE2 | IF2 RETURN | if (BOOL) : RETURN | if BOOL : RETURN | IF2 pass | if (BOOL) : pass | if BOOL : pass;
ELIF2	elif (BOOL) : START2 | elif BOOL : START2 | ELIF2 ELIF2 | ELIF2 ELSE2 | elif (BOOL) : RETURN | elif BOOL : RETURN;
ELSE2	else : START2 | else : RETURN;
START3	START3 START3 | VAR = CALC | VAR = VAL | VAR = ARRAY | VAR = NONE | VAR = FUNCTION | IF3 | FOR | WHILE | COMMENT | VAR + = VAL | VAR - = VAL | VAR * = VAL | VAR / = VAL | PRINT | RAISE | FUNCTION | WITH;
IF3	if (BOOL) : START3 | if BOOL : START3 | IF3 ELIF3 | IF3 ELSE3 | IF3 break | if (BOOL) : break | if BOOL : break | IF3 pass | if (BOOL) : pass | if BOOL : pass | if (BOOL) : continue | if BOOL : continue | IF3 continue;
ELIF3	elif (BOOL) : START3 | elif BOOL : START3 | ELIF3 ELIF3 | ELIF3 ELSE3 | elif (BOOL) : break | elif BOOL : break | ELIF3 break | elif (BOOL) : continue | elif BOOL : continue | ELIF3 continue | elif (BOOL) : pass | elif BOOL : pass | ELIF3 pass;
ELSE3	else : START3 | else : break | else : continue | else : pass;
PRINT	print (CONTENT) | print ( );
ARRAY	[CONTENT2] | [ARRAY] | ARRAY , ARRAY

CONTENT2	CONTENT2 , CONTENT2 | VAL | VAR | (CONTENT2) | WORD;
RAISE	raise VAR (CONTENT) | raise VAR ( )
FUNCTION	VAR (CONTENT2) | VAR ( )
WITH	with VAR as VAR : START

S = Start Symbol

START

2.2 Dekomposisi CNF

Berikut adalah CNF yang didapatkan dengan mengubah CFG di 2.1 menjadi CNF.

START	START START | VAR A1 | VAR B1 | VAR C1 | VAR D1 | VAR E1 | VAR F1 | VAR G1 | VAR H1 | VAR I1 | VAR J1 | VAR K1 | FQ Z1 | FQ AA1 | IF ELIF | IF ELSE | FD BG1 | FD BH1 | DEF RETURN | FD BI1 | FD BJ1 | FD BK1 | FD BL1 | DEF FK | FC BM1 | FC BN1 | FJ AM1 | FJ AN1 | FJ AO1 | FJ AP1 | FJ AQ1 | FN AE1 | FN AF1 | FN AG1 | FN AH1 | FN AI1 | FN AJ1 | FN AK1 | FN AL1 | WHILE FL | WHILE FK | WHILE FM | EZ VAR2 | EZ BP1 | FA BO1 | FG BE1 | FE BF1 | EX DU1 | EX DV1 | EU EB1 | EU EC1 | VAR ED1 | VAR EE1 | ET EF1
FZ	=
A1	FZ CALC
B1	FZ VAL
C1	FZ NONE
D1	FZ BOOL
E1	FZ CONTENT
F1	FZ ARRAY
G1	FZ FUNCTION
H1	OP H2
H2	FZ VAL
I1	OP I2

I2	FZ VAL
J1	OP J2
J2	FZ VAL
K1	OP K2
K2	FZ VAL
VAR	var
VAL	num | FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG
CONTENT	CONTENT CONTENT | FY L1 | num | var | VAL U1 | VAR W1 | VAL X1 | VAR Y1 | FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG | FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG
FY	(
FX	)
L1	CONTENT FX
BOOL	True | False | CONTENT M1 | CONTENT N1 | CONTENT O1 | CONTENT P1 | CONTENT Q1 | FS CONTENT | BOOL R1 | BOOL S1 | FS BOOL | FY T1
M1	BOOLOP CONTENT
FW	in
N1	FW CONTENT
FV	is
O1	FV CONTENT
FU	or
P1	FU CONTENT
FT	and
Q1	FT CONTENT
FS	not

R1	FU BOOL
S1	FT BOOL
T1	BOOL FX
BOOLOP	== | != | <= | >= | < | >
OP	+ | - | * | / | %
CALC	VAL U1 | VAR W1 | VAL X1 | VAR Y1
U1	OP VAL
W1	OP VAL
X1	OP VAR
Y1	OP VAR
FR	:
FQ	if
IF	FQ Z1 | FQ AA1 | IF ELIF | IF ELSE
Z1	FY Z2
Z2	BOOL Z3
Z3	FX Z4
Z4	FR START
AA1	BOOL AA2
AA2	FR START
FP	elif
ELIF	FP AB1 | FP AC1 | ELIF ELIF | ELIF ELSE
AB1	FY AB2
AB2	BOOL AB3

AB3	FX AB4
AB4	FR START
AC1	BOOL AC2
AC2	FR START
FO	else
ELSE	FO AD1
AD1	FR START
FN	while
WHILE	FN AE1 | FN AF1 | FN AG1 | FN AH1 | FN AI1 | FN AJ1 | FN AK1 | FN AL1 | WHILE FL | WHILE FK | WHILE FM
AE1	FY AE2
AE2	BOOL AE3
AE3	FX AE4
AE4	FR START3
AF1	BOOL AF2
AF2	FR START3
FM	break
AG1	FY AG2
AG2	BOOL AG3
AG3	FX AG4
AG4	FR FM
AH1	BOOL AH2
AH2	FR FM
FL	continue

AI1	FY AI2
AI2	BOOL AI3
AI3	FX AI4
AI4	FR FL
AJ1	BOOL AJ2
AJ2	FR FL
FK	pass
AK1	FY AK2
AK2	BOOL AK3
AK3	FX AK4
AK4	FR FK
AL1	BOOL AL2
AL2	FR FM
FJ	for
FOR	FJ AM1 | FJ AN1 | FJ AO1 | FJ AP1 | FJ AQ1
AM1	VAR AM2
AM2	FW AM3
AM3	VAR AM4
AM4	FR START3
AN1	VAR AN2
AN2	FW RANGE
AO1	VAR AO2
AO2	FW AO3

AO3	VAR AO4
AO4	FR FM
AP1	VAR AP2
AP2	FW AP3
AP3	VAR AP4
AP4	FR FK
AQ1	VAR AQ2
AQ2	FW AQ3
AQ3	VAR AQ4
AQ4	FR FL
FI	range
RANGE	FI AR1 | FI AS1 | FI AT1 | FI AU1 | FI AW1 | FI AX1 | FI AY1 | FI AZ1 | FI BA1 | FI BB1 | FI BC1 | FI BD1 | RANGE FL | RANGE FK | RANGE FM
AR1	FY AR2
AR2	VAL AR3
AR3	FX AR4
AR4	FR START3
FH	,
AS1	FY AS2
AS2	VAL AS3
AS3	FH AS4
AS4	VAL AS5
AS5	FX AS6
AS6	FR START3

AT1	FY AT2
AT2	VAL AT3
AT3	FH AT4
AT4	VAL AT5
AT5	FH AT6
AT6	VAL AT7
AT7	FX AT8
AT8	FR START3
AU1	FY AU2
AU2	VAL AU3
AU3	FX AU4
AU4	FR FM
AW1	FY AW2
AW2	VAL AW3
AW3	FH AW4
AW4	VAL AW5
AW5	FX AW6
AW6	FR FM
AX1	FY AX2
AX2	VAL AX3
AX3	FH AX4
AX4	VAL AX5
AX5	FH AX6

AX6	VAL AX7
AX7	FX AX8
AX8	FR FM
AY1	FY AY2
AY2	VAL AY3
AY3	FX AY4
AY4	FR FL
AZ1	FY AZ2
AZ2	VAL AZ3
AZ3	FH AZ4
AZ4	VAL AZ5
AZ5	FX AZ6
AZ6	FR FL
BA1	FY BA2
BA2	VAL BA3
BA3	FH BA4
BA4	VAL BA5
BA5	FH BA6
BA6	VAL BA7
BA7	FX BA8
BA8	FR FL
BB1	FY BB2
BB2	VAL BB3

BB3	FX BB4
BB4	FR FK
BC1	FY BC2
BC2	VAL BC3
BC3	FH BC4
BC4	VAL BC5
BC5	FX BC6
BC6	FR FK
BD1	FY BD2
BD2	VAL BD3
BD3	FH BD4
BD4	VAL BD5
BD5	FH BD6
BD6	VAL BD7
BD7	FX BD8
BD8	FR FK
FG	'
FF	word
COMMENT	FG BE1 | FE BF1
BE1	FG BE2
BE2	FG BE3
BE3	FF BE4
BE4	FG BE5

BE5	FG FG
FE	"
BF1	FE BF2
BF2	FE BF3
BF3	FF BF4
BF4	FE BF5
BF5	FE FE
FD	def
DEF	FD BG1 | FD BH1 | DEF RETURN | FD BI1 | FD BJ1 | FD BK1 | FD BL1 | DEF FK
BG1	VAR BG2
BG2	FY BG3
BG3	CONTENT BG4
BG4	FX BG5
BG5	FR START2
BH1	VAR BH2
BH2	FY BH3
BH3	FX BH4
BH4	FR START2
BI1	VAR BI2
BI2	FY BI3
BI3	CONTENT BI4
BI4	FX BI5
BI5	FR RETURN

BJ1	VAR BJ2
BJ2	FY BJ3
BJ3	FX BJ4
BJ4	FR RETURN
BK1	VAR BK2
BK2	FY BK3
BK3	CONTENT BK4
BK4	FX BK5
BK5	FR FK
BL1	VAR BL2
BL2	FY BL3
BL3	FX BL4
BL4	FR FK
FC	class
CLASS	FC BM1 | FC BN1
BM1	VAR BM2
BM2	FR START
BN1	VAR BN2
BN2	FR FK
FB	return
RETURN	FB BOOL | FB VAL | FB WORD
FA	from
FROM	FA BO1

BO1	VAR IMPORT
EZ	import
IMPORT	EZ VAR2 | EZ BP1
EY	as
BP1	VAR BP2
BP2	EY VAR
VAR2	VAR2 BQ1 | var
BQ1	FH VAR2
WORD	FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG
BR1	FF FE
BS1	FF FG
BT1	OP WORD
START2	START2 START2 | VAR BU1 | VAR BW1 | VAR BX1 | VAR BY1 | VAR BZ1 | VAR CA1 | VAR CB1 | VAR CC1 | VAR CD1 | FQ CE1 | FQ CF1 | IF2 ELIF2 | IF2 ELSE2 | IF2 RETURN | FQ CG1 | FQ CH1 | IF2 FK | FQ CI1 | FQ CJ1 | FJ AM1 | FJ AN1 | FJ AO1 | FJ AP1 | FJ AQ1 | FN AE1 | FN AF1 | FN AG1 | FN AH1 | FN AI1 | FN AJ1 | FN AK1 | FN AL1 | WHILE FL | WHILE FK | WHILE FM | FG BE1 | FE BF1 | EX DU1 | EX DV1 | EU EB1 | EU EC1 | VAR ED1 | VAR EE1 | ET EF1
BU1	FZ CALC
BW1	FZ VAL
BX1	FZ ARRAY
BY1	FZ NONE
BZ1	FZ FUNCTION
CA1	OP CA2
CA2	FZ VAL

CB1	OP CB2
CB2	FZ VAL
CC1	OP CC2
CC2	FZ VAL
CD1	OP CD2
CD2	FZ VAL
IF2	FQ CE1 | FQ CF1 | IF2 ELIF2 | IF2 ELSE2 | IF2 RETURN | FQ CG1 | FQ CH1 | IF2 FK | FQ CI1 | FQ CJ1
CE1	FY CE2
CE2	BOOL CE3
CE3	FX CE4
CE4	FR START2
CF1	BOOL CF2
CF2	FR START2
CG1	FY CG2
CG2	BOOL CG3
CG3	FX CG4
CG4	FR RETURN
CH1	BOOL CH2
CH2	FR RETURN
CI1	FY CI2
CI2	BOOL CI3
CI3	FX CI4
CI4	FR FK

CJ1	BOOL CJ2
CJ2	FR FK
ELIF2	FP CK1 | FP CL1 | ELIF2 ELIF2 | ELIF2 ELSE2 | FP CM1 | FP CN1
CK1	FY CK2
CK2	BOOL CK3
CK3	FX CK4
CK4	FR START2
CL1	BOOL CL2
CL2	FR START2
CM1	FY CM2
CM2	BOOL CM3
CM3	FX CM4
CM4	FR RETURN
CN1	BOOL CN2
CN2	FR RETURN
ELSE2	FO CO1 | FO CP1
CO1	FR START2
CP1	FR RETURN
START3	START3 START3 | VAR CQ1 | VAR CR1 | VAR CS1 | VAR CT1 | VAR CU1 | VAR CW1 | VAR CX1 | VAR CY1 | VAR CZ1 | FQ DA1 | FQ DB1 | IF3 ELIF3 | IF3 ELSE3 | IF3 FM | FQ DC1 | FQ DD1 | IF3 FK | FQ DE1 | FQ DF1 | FQ DG1 | FQ DH1 | IF3 FL | FJ AM1 | FJ AN1 | FJ AO1 | FJ AP1 | FJ AQ1 | FN AE1 | FN AF1 | FN AG1 | FN AH1 | FN AI1 | FN AJ1 | FN AK1 | FN AL1 | WHILE FL | WHILE FK | WHILE FM | FG BE1 | FE BF1 | EX DU1 | EX DV1 | EU EB1 | EU EC1 | VAR ED1 | VAR EE1 | ET EF1
CQ1	FZ CALC

CR1	FZ VAL
CS1	FZ ARRAY
CT1	FZ NONE
CU1	FZ FUNCTION
CW1	OP CW2
CW2	FZ VAL
CX1	OP CX2
CX2	FZ VAL
CY1	OP CY2
CY2	FZ VAL
CZ1	OP CZ2
CZ2	FZ VAL
IF3	FQ DA1 | FQ DB1 | IF3 ELIF3 | IF3 ELSE3 | IF3 FM | FQ DC1 | FQ DD1 | IF3 FK | FQ DE1 | FQ DF1 | FQ DG1 | FQ DH1 | IF3 FL
DA1	FY DA2
DA2	BOOL DA3
DA3	FX DA4
DA4	FR START3
DB1	BOOL DB2
DB2	FR START3
DC1	FY DC2
DC2	BOOL DC3
DC3	FX DC4
DC4	FR FM

DD1	BOOL DD2
DD2	FR FM
DE1	FY DE2
DE2	BOOL DE3
DE3	FX DE4
DE4	FR FK
DF1	BOOL DF2
DF2	FR FK
DG1	FY DG2
DG2	BOOL DG3
DG3	FX DG4
DG4	FR FL
DH1	BOOL DH2
DH2	FR FL
ELIF3	FP DI1 | FP DJ1 | ELIF3 ELIF3 | ELIF3 ELSE3 | FP DK1 | FP DL1 | ELIF3 FM | FP DM1 | FP DN1 | ELIF3 FL | FP DO1 | FP DP1 | ELIF3 FK
DI1	FY DI2
DI2	BOOL DI3
DI3	FX DI4
DI4	FR START3
DJ1	BOOL DJ2
DJ2	FR START3
DK1	FY DK2
DK2	BOOL DK3

DK3	FX DK4
DK4	FR FM
DL1	BOOL DL2
DL2	FR FM
DM1	FY DM2
DM2	BOOL DM3
DM3	FX DM4
DM4	FR FL
DN1	BOOL DN2
DN2	FR FL
DO1	FY DO2
DO2	BOOL DO3
DO3	FX DO4
DO4	FR FK
DP1	BOOL DP2
DP2	FR FK
ELSE3	FO DQ1 | FO DR1 | FO DS1 | FO DT1
DQ1	FR START3
DR1	FR FM
DS1	FR FL
DT1	FR FK
EX	print
PRINT	EX DU1 | EX DV1

DU1	FY DU2
DU2	CONTENT FX
DV1	FY FX
EW	[
EV	]
ARRAY	EW DW1 | EW DX1 | ARRAY DY1
DW1	CONTENT2 EV
DX1	ARRAY EV
DY1	FH ARRAY
CONTENT 2	CONTENT2 DZ1 | FY EA1 | num | var | FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG | FE BR1 | FG BS1 | WORD BT1 | FE FE | FG FG
DZ1	FH CONTENT2
EA1	CONTENT2 FX
EU	raise
RAISE	EU EB1 | EU EC1
EB1	VAR EB2
EB2	FY EB3
EB3	CONTENT FX
EC1	VAR EC2
EC2	FY FX
FUNCTIO N	VAR ED1 | VAR EE1
ED1	FY ED2
ED2	CONTENT2 FX

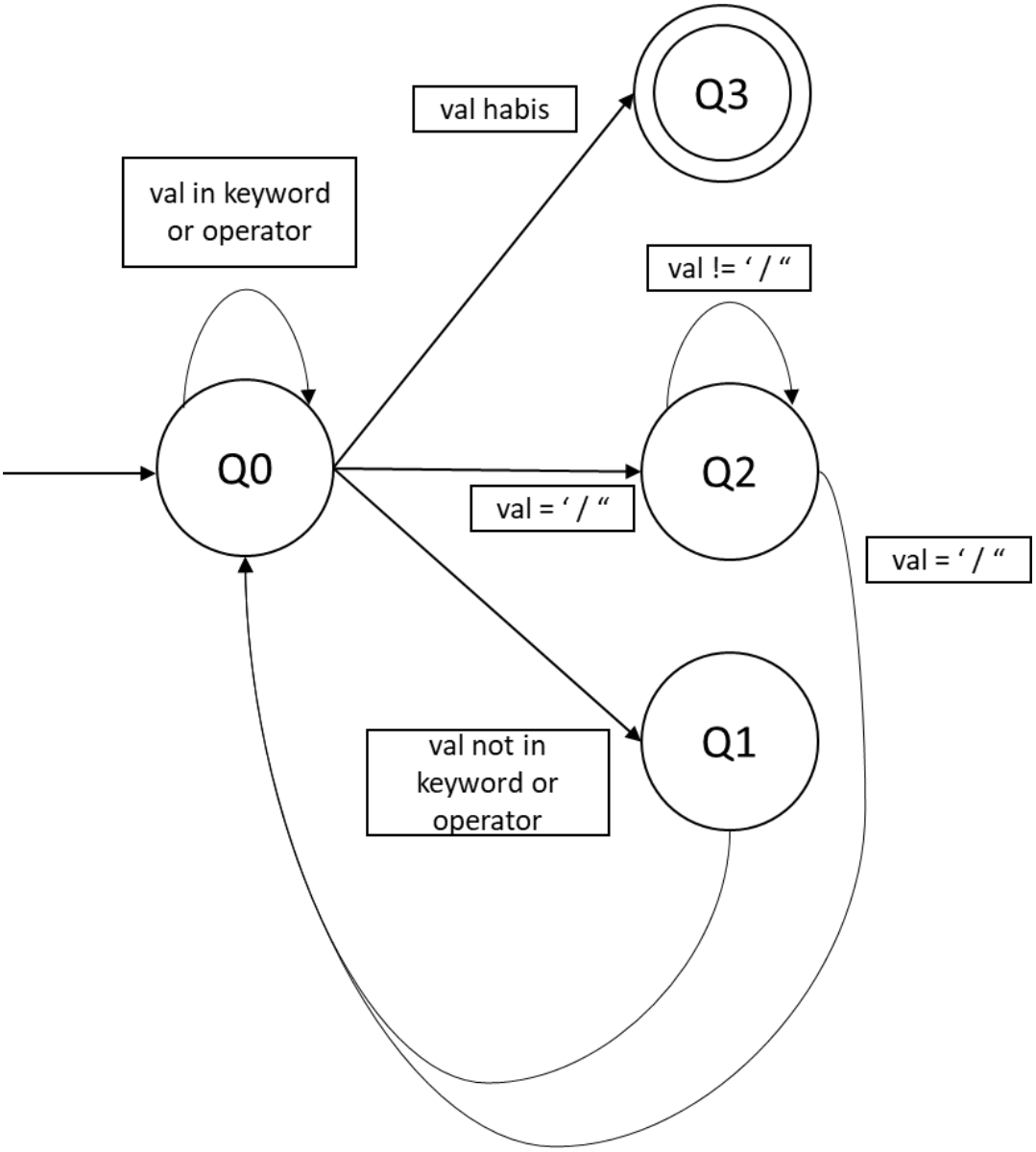
EE1	FY FX
ET	with
WITH	ET EF1
EF1	VAR EF2
EF2	EY EF3
EF3	VAR EF4
EF4	FR START
S0	START START | VAR A1 | VAR B1 | VAR C1 | VAR D1 | VAR E1 | VAR F1 | VAR G1 | VAR H1 | VAR I1 | VAR J1 | VAR K1 | FQ Z1 | FQ AA1 | IF ELIF | IF ELSE | FD BG1 | FD BH1 | DEF RETURN | FD BI1 | FD BJ1 | FD BK1 | FD BL1 | DEF FK | FC BM1 | FC BN1 | FJ AM1 | FJ AN1 | FJ AO1 | FJ AP1 | FJ AQ1 | FN AE1 | FN AF1 | FN AG1 | FN AH1 | FN AI1 | FN AJ1 | FN AK1 | FN AL1 | WHILE FL | WHILE FK | WHILE FM | EZ VAR2 | EZ BP1 | FA BO1 | FG BE1 | FE BF1 | EX DU1 | EX DV1 | EU EB1 | EU EC1 | VAR ED1 | VAR EE1 | ET EF1

2.3 Dekomposisi FA

Untuk mengecek validitas variabel, menentukan jika suatu input adalah variabel, string, atau integer, isi file input harus diperiksa dan dimanipulasi menggunakan Finite Automata agar dapat diperiksa dengan benar.

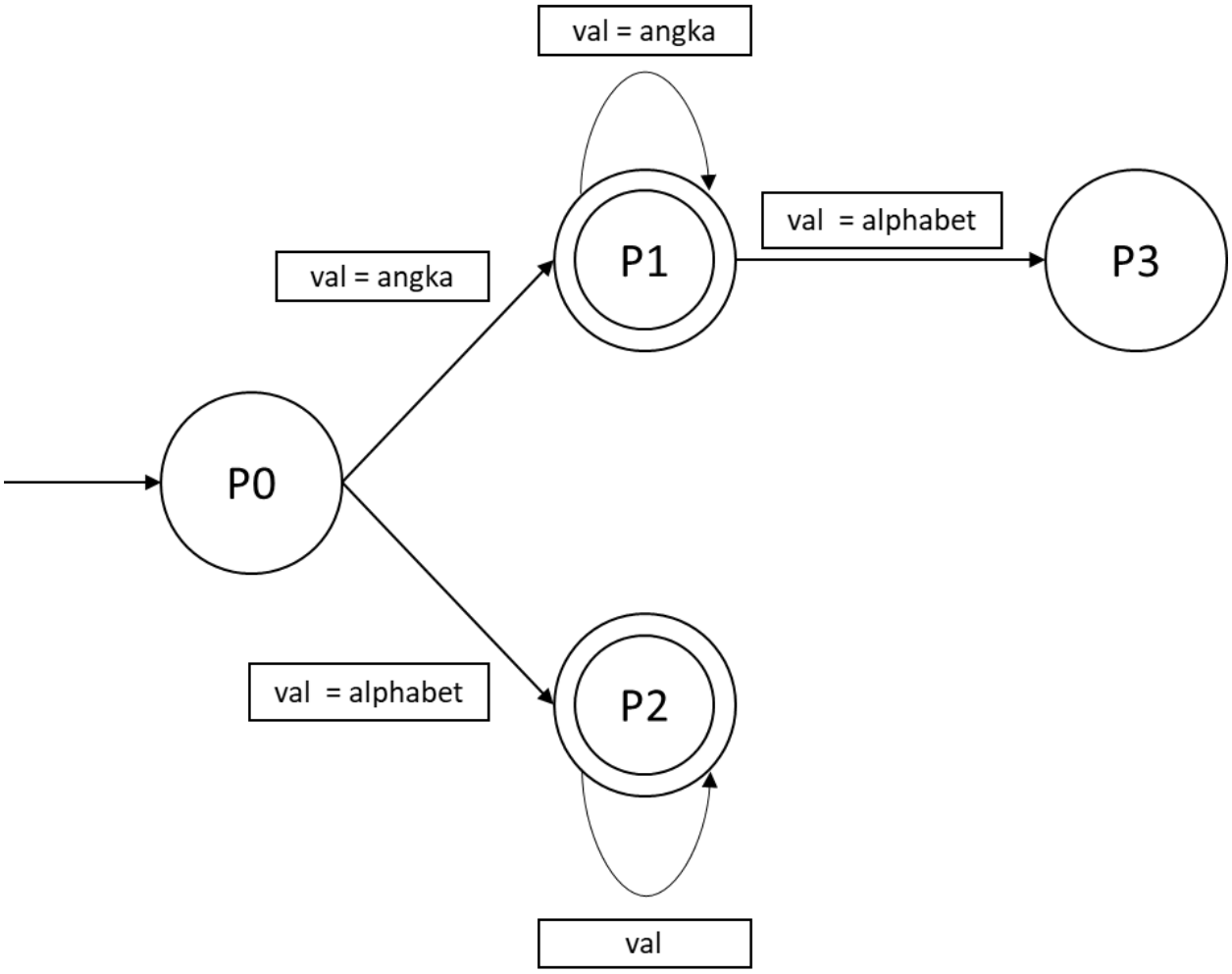
Pertama diperlukan data yang digunakan untuk menentukan jika bagian dalam file adalah sebuah variabel atau bukan dengan mencatat nilai khusus. Nilai khusus yang digunakan dibagi menjadi dua yaitu keyword dan operator. Keyword dan operator yang disimpan dapat dilihat pada bagian terminal di dekomposisi CFG. Program juga harus dapat mencatat jika nilai tersebut adalah sebuah angka atau huruf sehingga dibuat array dengan isi angka 1 - 9 dan array berisi semua huruf dan huruf kapital a sampai z.

Jika kita misalkan “val” sebagai isi array hasil pengolahan file secara terurut dan membuat sebuah finite automata, maka dapat digambarkan sebagai berikut :



- Q0 menunjukkan awal mulai pembacaan array dan val adalah nilai dari array secara terurut. Jika val ada dalam catatan keyword atau operator maka val tidak akan diolah dan program membaca bagian array selanjutnya.
- Q1 menunjukkan bahwa val tidak ada dalam catatan keyword atau operator sehingga val mungkin merupakan variabel atau integer. Di dalam Q1 akan diproses lagi dan nilai val akan diubah lalu kembali ke Q0 dimana program membaca bagian array selanjutnya.
- Q2 menunjukkan bahwa ada string di dalam file sehingga program mengubah setiap val antara 2 tanda petik. Setelah tanda petik ke-2 ditemukan, program kembali ke Q0 dan program membaca bagian array selanjutnya.
- Q3 menunjukkan bahwa semua isi dalam array sudah dibaca dan program berhenti.

Jika ingin dibuat finite automata untuk proses dalam Q1 dengan “val” sebagai karakter dalam input untuk Q1 maka dapat dibuat sebagai berikut :



- P0 menunjukkan awal program mulai dan menerima input dalam bentuk kata yang tidak ada dalam keyword atau operator
- P1 menunjukkan jika karakter pertama dari input adalah angka sehingga nilai array tersebut diubah menjadi “num” yang menunjukkan angka.
- P2 menunjukkan jika karakter pertama dari input adalah huruf sehingga nilai array tersebut diubah menjadi “var” yang menunjukkan variabel
- P3 menunjukkan jika ada huruf setelah angka (contoh : 123a) sehingga nilai array bukan angka atau variabel.

BAB III
Implementasi dan Pengujian

3.1 Spesifikasi Kode Program

Pada implementasi, program menggunakan beberapa file yang berbeda, beberapa file yang berbeda ini bisa dipanggil dengan menggunakan module import pada python. Beberapa file tersebut adalah.

1. CFG2CNF.py

File ini digunakan untuk mengonversi CFG menjadi CNF. File ini, diambil dari https://github.com/adelmassimo/CFG2CNF. Program akan mengonversi CFG yang dimasukkan menjadi CNF dengan keluaran CNF dari program berupa out.txt. Program ini memiliki beberapa fungsi dan prosedur di dalamnya. Fungsi dan prosedure tersebut adalah.

No	Fungsi / Prosedur	Keterangan
1	isUnitary	Mengecek apakah suatu rule tepat memiliki production tepatnya 1 simbol terminal dan non-terminal untuk menjadi bagian dari bagian variabel
2	isSimple	Mengecek apakah suatu rule tepat memiliki production tepatnya 1 simbol terminal dan non-terminal untuk menjadi bagian dari bagian variabel V
3	START	Menambahkan suatu start symbol baru bernama S0 ke dalam daftar variabel dan rules yang telah ada
3	TERM	Menghapus rule yang mengandung terminal dan non-terminal simbol dan mengubahnya ke bentuk 2 non-terminal atau 1 terminal
4	BIN	Fungsi untuk mengeliminasi non-unitry rule
6	DEL	Fungsi untuk melakukan penghapusan non-terminal rule
7	unit_routin	Memeriksa apakah suatu unit atau rules sudah berbentuk unary
8	UNIT	Mengeliminasi unit production dalam suatu rule

2. helper.py

File helper.py berisi prosedur dan fungsi untuk melakukan pembersihan terhadap production dari CFG sebelum diubah ke CNF. File helper.py ini merupakan penunjang dari CFG2CNF.py. File ini, diambil dari https://github.com/adelmassimo/CFG2CNF. File ini juga berisi beberapa fungsi dan prosedure. Fungsi dan prosedure tersebut adalah.

No	Fungsi / Prosedur	Keterangan
1	union	Untuk menggabungkan 2 list
2	loadModel	Memuat model dari CFG dan dibagi berdasarkan Terminal,Production, dan Variable
3	cleanProduction	Melakukan pembersihan pada production dan dimuat dalam bentuk list agar dapat diproses
4	cleanAlphabet	Melakukan pembersihan pada terminal dan variabel dan dimuat dalam bentuk list agar dapat diproses
5	seekAndDestroy	Melakukan eliminasi terhadap variabel yang tidak diperlukan
6	setupDict	Melakukan penelusuran terhadap unit variabel
7	rewrite	Melakukan pembacaan CFG dengan format tertentu menjadi format yang bisa dibaca satu per satu dalam array
8	dict2set	Memasukkan dictionary ke dalam suatu list atau set
9	pprintRule	Menampilkan hasil dari rules
10	prettyForm	Melakukan penggabungan untuk production yang memiliki lebih dari 1 hasil

3. Checker1.py

File checker1.py memanggil prosedur dari file CNFtoDict, Lexer_1, dan simpanparse untuk memeriksa sebuah input file .py dan mengeluarkan hasil “Accepted” jika tidak ditemukan kesalahan atau “Syntax Error” jika ditemukan kesalahan. Checker1 menggabungkan semua isi input file ke sebuah array dan langsung diproses.

4. Checker2.py

File checker1.py memanggil prosedur dari file CNFtoDict, Lexer_2, dan simpanparse2 untuk memeriksa sebuah input file .py dan mengeluarkan hasil “Accepted” jika tidak ditemukan kesalahan atau “Syntax Error” jika ditemukan kesalahan. Checker2 memeriksa input file per baris.

5. simpanparse.py

File simpanparse.py berisi fungsi cykParse. Fungsi cykParse menerima input “w” yang merupakan sebuah input file yang sudah dipecah menjadi array serta variabel, angka, dan perintah lainnya sudah diolah oleh fungsi lain lalu menerima input ”R” yang merupakan CNF yang sudah diolah sehingga dapat digunakan dalam Python. Di dalam cykParse terdapat algoritma CYK yang digunakan untuk melihat jika ada kesalahan syntax dalam file. Fungsi simpanparse digunakan dalam checker1 dan mengolah seluruh isi input file.

6. simpanparse2.py

File simpanparse.py berisi fungsi cykParse dan fungsi parseAkhir. Fungsi cykParse menerima input “w” yang merupakan sebuah input file yang sudah dipecah menjadi array serta variabel, angka, dan perintah lainnya sudah diolah oleh fungsi lain lalu menerima input ”R” yang merupakan CNF yang sudah diolah sehingga dapat digunakan dalam Python. Di dalam cykParse terdapat algoritma CYK yang digunakan untuk melihat jika ada kesalahan syntax dalam file. Fungsi parseAkhir menerima input “w” yang merupakan matriks hasil dari pengolahan input file dan “R” yang merupakan CNF seperti pada fungsi cykParse. Fungsi parseAkhir membaca input file per line sesuai dengan hasil matriks dan mengolah hasil pembacaan tersebut menggunakan fungsi cykParse.

7. lexer_1.py

File lexer_1.py berisi fungsi lexer yang digunakan untuk mengolah input file. Semua isi input file akan dimasukkan ke sebuah array dan dipecah berdasarkan keberadaan spasi, operator dan jika bagian file tersebut adalah perintah. Setelah itu, array akan diperiksa ulang dan dimanipulasi lebih lanjut sehingga dapat diperiksa dengan algoritma CYK.

8. lexer_2.py

File lexer_1.py berisi fungsi lexer yang digunakan untuk mengolah input file. Semua isi input file akan dimasukkan ke sebuah matriks dan dipecah berdasarkan keberadaan spasi, newline, operator, dan jika bagian file tersebut adalah perintah. Setelah

itu, matriks akan diperiksa ulang dan dimanipulasi lebih lanjut sehingga dapat diperiksa dengan algoritma CYK.

9. CNFtoDict.py

File CNFtoDict.py berisi fungsi CNFtoDict mengubah file cnf yang dihasilkan dari CFG2CNF.py menjadi sebuah dictionary dalam bahasa Python. Dictionary tersebut akan digunakan dalam fungsi lain untuk memeriksa input file.

3.2 *Test Case* 1

Program akan diuji menggunakan file input1.py dengan isi file sebagai berikut.

```
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 "Dododo"
```

Dengan menggunakan compiler yang dibuat, didapatkan hasil :

```
PS E:\tubes tbfo\tubes-tbfo> python checker1.py input1.py
Accepted
PS E:\tubes tbfo\tubes-tbfo>
```

Hasil pengujian dari compiler menghasilkan *Accepted*, artinya pada file input1.py tidak ada *syntax* yang salah.

3.3 *Test Case 2*

Program akan diuji menggunakan file input2.py dengan isi file sebagai berikut.

```
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 "dododo"
```

Dengan menggunakan compiler yang dibuat, didapatkan hasil :

```
PS E:\tubes tbfo\tubes-tbfo> python checker1.py input2.py
Syntax Error
PS E:\tubes tbfo\tubes-tbfo>
```

Hasil pengujian dari compiler menghasilkan *Syntax Error*, artinya pada file input2.py ada *syntax* yang error.

3.4 *Test Case 3*

Program akan diuji menggunakan file input3.py dengan isi file sebagai berikut.

```
for i in range(5):
    print("haloo")
    if i == 4:
        break
```

Dengan menggunakan compiler yang dibuat didapatkan hasil :

```
PS E:\tubes tbfo\tubes-tbfo> python checker1.py input3.py
break
Accepted
PS E:\tubes tbfo\tubes-tbfo>
```

Hasil pengujian dari compiler menghasilkan *Accepted*, artinya pada file input3.py tidak ada *syntax* yang salah.

BAB IV
Kesimpulan dan Saran

4.1 Kesimpulan

Program compiler yang dibuat berjalan dengan cukup baik mengingat masih ada beberapa bug atau error yang terjadi pada saat menjalankan program. Walaupun ada beberapa bug atau error, program mampu menampung beberapa kasus kecil, namun untuk kasus yang lebih besar kompleks masih ada beberapa bug atau error yang terjadi, seperti hasil yang tidak sesuai dengan yang seharusnya dan waktu compile yang cukup lama. Hal tersebut terjadi karena CFG yang digunakan pada program tidak bisa menampung semua kasus yang ada pada file yang di inputkan pada program.

4.2 Saran

Berikut adalah saran yang bisa diberikan dalam pengerjaan tugas ini :

1. Banyak mencari referensi *Context Free Grammar* terkait dengan compiler python, sehingga program bisa menampung semua *test case.*
2. Lebih banyak mencari informasi mengenai parsing dan lexer atau tokenization.
3. Menggunakan CFG dan CNF yang lebih rapi dan efisien.
4. Melakukan pemeriksaan file input dengan algoritma yang lebih efisien.

Lampiran

Link github kelompok : https://github.com/wmk567/tubes-tbfo

Referensi :

- Adelmassimo, “CFG2CNF”, https://github.com/adelmassimo/CFG2CNF
- https://www.geeksforgeeks.org/cyk-algorithm-for-context-free-grammar/
- https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/

Pembagian kerja dalam kelompok :

No	NIM	Nama	Pembagian Kerja
1	13520020	Willian Manuel Kurniawan	• CYK, CFG, CNFtoDict
2	13520032	Fadil Fauzani	• Lexer / tokenizer
2	13520041	Ilham Pratama	• CFG, Laporan