

# **COMPILER BAHASA PYTHON**

## **LAPORAN TUGAS BESAR**

Diajukan Untuk Memenuhi Tugas Teori Bahasa Formal dan Otomata



oleh

**KWETIAUW AWIH HALAL**

**MATTHEW KEVIN AMADEUS                      13518035**

**MICHAEL HANS                                      13518056**

**LIONNARTA SAVIRANDY                      13518128**

**TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2019**

# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB I</b>	
<b>DASAR TEORI</b>	<b>2</b>
1.1 Context-Free Grammar	2
Parsing	3
Ambiguitas	4
1.2 Chomsky Normal Form	5
Transformasi CFG ke CNF	5
Pembentukan Normal Chomsky Form	6
1.3 Cocke-Younger Kasami	8
1.4 Bahasa Pemrograman Python	9
<b>BAB II</b>	
<b>ANALISIS PERSOALAN DAN DEKOMPOSISI</b>	<b>10</b>
2.1 CFG untuk Python	10
2.2 CNF untuk Python	12
<b>BAB III</b>	
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>18</b>
3.1 Spesifikasi Teknis Program	18
<b>BAB IV</b>	
<b>EKSPERIMEN</b>	<b>23</b>
4.1 Tampilan Awal	23
4.2 Percabangan (If, Elif, Else)	24
4.3 Looping	26
4.4 Class, Import, Def	27
4.5 All in One	28
<b>BAB V</b>	
<b>PENUTUP</b>	<b>31</b>
5.1 Kesimpulan	31
5.2 Saran	31
<b>REFERENSI</b>	<b>32</b>

# BAB I

## DASAR TEORI

### 1.1 CONTEXT-FREE GRAMMAR

*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.

Definisi formal dari CFG dapat didefinisikan sebagai berikut.

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

V = himpunan terbatas variabel

T = himpunan terbatas terminal

P = himpunan terbatas dari produksi

S = start symbol

Context Free Grammar (CFG)/ Bahasa Bebas Konteks adalah sebuah tata bahasa dimana tidak terdapat pembatasan pada hasil produksinya. Contoh pada aturan produksi :

$$\alpha \rightarrow \beta$$

batasannya hanyalah ruas kiri ( $\alpha$ ) adalah sebuah simbol variabel. Sedangkan contoh aturan produksi yang termasuk CFG adalah seperti di bawah :

- $B \rightarrow CDeFg$
- $D \rightarrow BcDe$

Proses parsing adalah proses pembacaan string dalam bahasa sesuai CFG tertentu, proses ini harus mematuhi aturan produksi dalam CFG tersebut

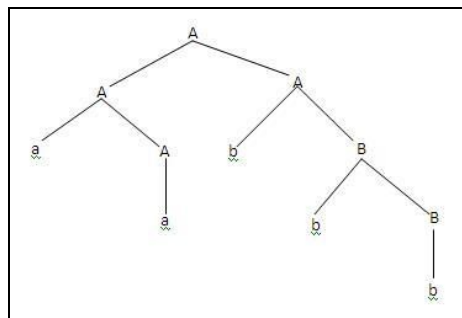
## Parsing

Context Free Grammar ( CFG ) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Pohon penurunan ( *derivation tree/parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan.

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

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

Maka jika ingin dicari gambar *pohon penurunan* dengan string : ‘aabb’ hasilnya adalah seperti di bawah :



Gambar 1.1 Contoh pembacaan string ‘aabb’

## Context Free Grammar (CFG) - Parse Tree

Proses penurunan / parsing bisa dilakukan dengan cara sebagai berikut :

- Penurunan terkiri (*leftmost derivation*): simbol variabel terkiri yang di perluas terlebih dahulu.
- Penurunan terkanan ( *rightmost derivation* ) : simbol variabel terkanan yang diperluas terlebih dahulu.

Misal : Grammar sebagai berikut :

- $S \rightarrow aAS \mid a$
- $A \rightarrow SbA \mid ba$

Untuk memperoleh string 'aabbaa' dari grammar diatas dilakukan dengan cara :

- Penurunan terkiri:  $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$
- Penurunan terkanan :  $S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aAbbaa \Rightarrow aabbaa$

## Ambiguitas

*Ambiguitas* terjadi bila terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu string.

Misalkan terdapat tata bahasa sebagai berikut :

- $S \rightarrow A \mid B$
- $A \rightarrow a$
- $B \rightarrow a$

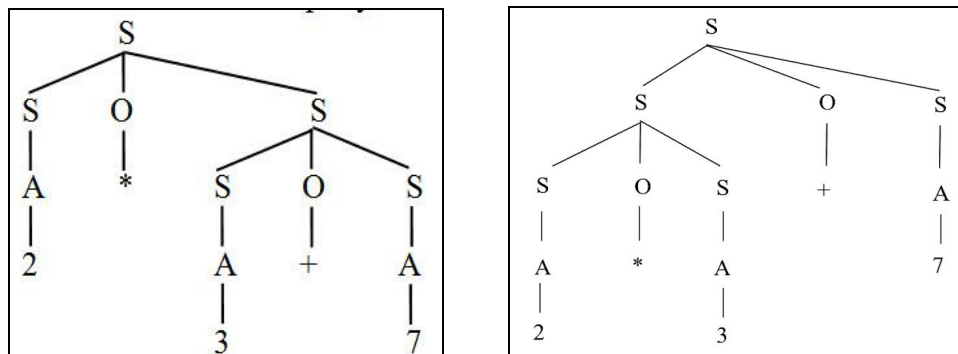
Untuk memperoleh untai 'a' bisa terdapat dua cara penurunan sebagai berikut :

- $S \Rightarrow A \Rightarrow a$
- $S \Rightarrow B \Rightarrow a$

Contoh ambiguitas lain:

Diketahui grammar  $G = \{S \rightarrow SOS \mid A, O \rightarrow * \mid +, A \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9\}$

String :  $2*3+7$  mempunyai dua pohon sintaks berikut :



Gambar 1.2 Contoh Ambiguitas Grammar

Sebuah string yang mempunyai lebih dari satu pohon sintaks disebut *string ambigu(ambiguous)*. Grammar yang menghasilkan paling sedikit sebuah string ambigu disebut *grammar ambigu*.

## 1.2 CHOMSKY NORMAL-FORM

Bentuk normal Chomsky / *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  $\epsilon$ . Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi *bentuk normal Chomsky* dengan syarat tata bahasa bebas konteks tersebut:

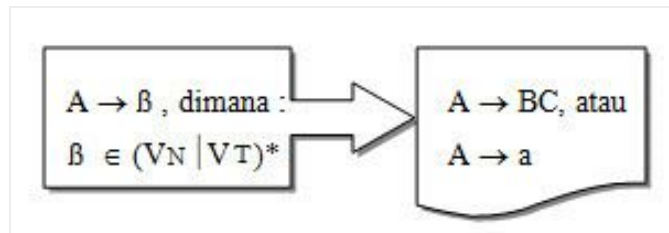
- Tidak memiliki produksi *useless*
- Tidak memiliki produksi unit
- Tidak memiliki produksi  $\epsilon$

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

$$A \rightarrow BC \text{ atau } A \rightarrow a.$$

### TRANSFORMASI CFG KE CNF

Transformasi CFG ke CNF adalah transformasi berikut :



Gambar 1.3 Transformasi CFG ke CNF

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

Misalkan

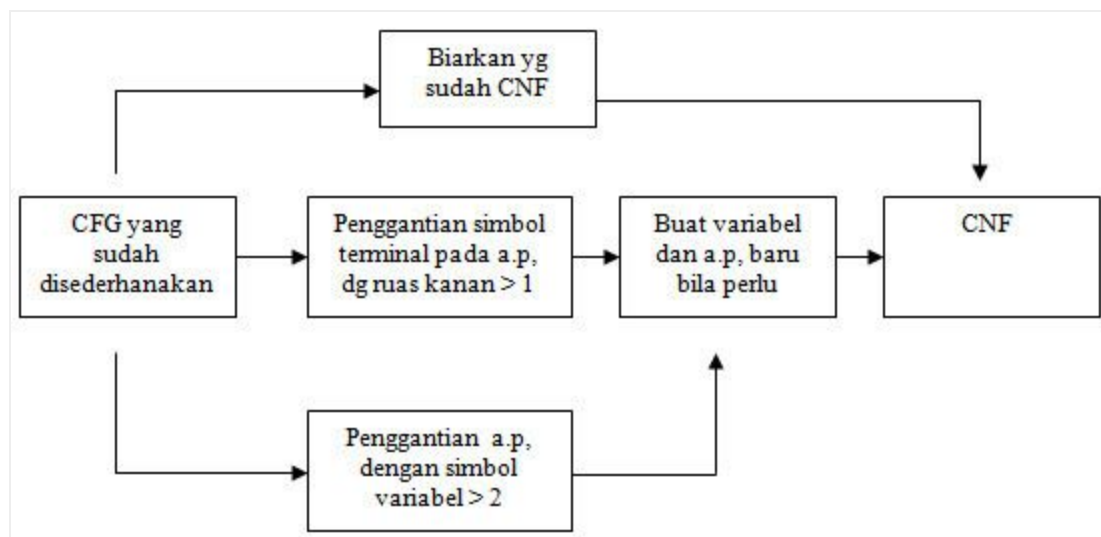
- $A \rightarrow BC$
- $A \rightarrow b$
- $B \rightarrow a$
- $C \rightarrow BA \mid d$

## PEMBENTUKAN *BENTUK NORMAL CHOMSKY*

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

- Biarkan aturan produksi yang sudah dalam *bentuk normal Chomsky*
- Lakukan penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan panjang ruas kanan  $> 1$
- Lakukan penggantian aturan produksi yang ruas kanannya memuat  $> 2$  simbol variabel
- Penggantian-penggantian tersebut bisa dilakukan berkali-kali sampai akhirnya semua aturan produksi dalam *bentuk normal Chomsky*
- 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:



Gambar 1.4 Langkah-langkah pembentukan bentuk normal Chomsky

**Contoh :** Diberikan Context-Free Grammar (CFG) sebagai berikut.

$$S \rightarrow aB \mid CA$$

$$B \rightarrow BC \mid Ab$$

$$A \rightarrow a \mid bc$$

$$C \rightarrow aB \mid b$$

Aturan produksi yang sudah dalam *bentuk normal Chomsky* :

$$S \rightarrow CA$$

$$B \rightarrow BC$$

$$A \rightarrow a$$

$$C \rightarrow b$$

Penggantian aturan produksi yang belum dalam *bentuk normal Chomsky*:

$$S \rightarrow aB \Rightarrow S \rightarrow P1B$$

$$B \rightarrow Ab \Rightarrow B \rightarrow A P2$$

$$A \rightarrow bc \Rightarrow A \rightarrow P2P3$$

$$C \rightarrow aB \Rightarrow C \rightarrow P1B$$

Terbentuk aturan produksi dan simbol variabel baru:

$$P1 \rightarrow a$$

$$P2 \rightarrow b$$

$$P3 \rightarrow c$$

Hasil akhir aturan produksi dalam *bentuk normal Chomsky*

$$S \rightarrow CA$$

$$S \rightarrow P1B$$

$$P1 \rightarrow a$$

$$A \rightarrow a$$

$$S \rightarrow P2P3$$

$$P2 \rightarrow b$$

$$B \rightarrow BC$$

$$B \rightarrow A P2$$

$$P3 \rightarrow c$$

$$C \rightarrow b$$

$$C \rightarrow P1B$$

Misalkan  $P1 = D$ ,  $P2 = E$ ,  $P3 = F$ , maka aturan produksinya menjadi:

$$S \rightarrow CA$$

$$S \rightarrow DB$$

$$D \rightarrow a$$

$$A \rightarrow a$$

$$S \rightarrow EF$$

$$E \rightarrow b$$

$$B \rightarrow BC$$

$$B \rightarrow AE$$

$$F \rightarrow c$$

$$C \rightarrow b$$

$$C \rightarrow DB$$



### 1.3 COCKE-YOUNGER KASAMI

Cocke-Younger-Kasami (CYK) Algorithm adalah salah satu algoritma parsing untuk Context-Free Grammar yang sudah dikonversi ke bentuk Chomsky Normal Form. Tujuan algoritma CYK ini adalah untuk membership testing, atau menunjukkan apakah suatu string dapat diterima atau bagian dari language CFG tersebut.

CYK-Algorithm dapat diilustrasikan dalam bentuk tabel sebagai berikut. Berikut merupakan tabel yang dibentuk jika string yang akan dicek adalah string dengan panjang 5.

X <sub>15</sub>				
X <sub>14</sub>	X <sub>25</sub>			
X <sub>13</sub>	X <sub>24</sub>	X <sub>35</sub>		
X <sub>12</sub>	X <sub>23</sub>	X <sub>34</sub>	X <sub>45</sub>	
X <sub>11</sub>	X <sub>22</sub>	X <sub>33</sub>	X <sub>44</sub>	X <sub>55</sub>
b	a	a	b	a

Dari tabel tersebut, X<sub>15</sub> adalah akar sehingga bila akar mengandung start symbol, maka string merupakan bagian dari Language L atau string diterima oleh language L tersebut.

Contoh :

Misalkan suatu CFG didefinisikan sebagai

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

G mempunyai produksi :

$$S \rightarrow AB \mid BC$$

$$A \rightarrow BA \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow AB \mid a$$

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 karena akar mengandung start symbol S.

## 1.4 BAHASA PEMROGRAMAN PYTHON

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python juga didukung oleh komunitas yang besar.

Python mendukung multi paradigma pemrograman, utamanya; namun tidak dibatasi; pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, python umumnya digunakan sebagai bahasa skrip meski pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa skrip. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. Saat ini kode python dapat dijalankan di berbagai platform sistem operasi, beberapa di antaranya adalah Linux/Unix, Windows, Mac OS/X, Java Virtual Machine, Amiga, Palm, Symbian.

Beberapa fitur yang dimiliki Python adalah:

- memiliki kepustakaan yang luas; dalam distribusi Python telah disediakan modul-modul 'siap pakai' untuk berbagai keperluan.
- memiliki tata bahasa yang jernih dan mudah dipelajari.
- memiliki aturan layout kode sumber yang memudahkan pengecekan, pembacaan kembali dan penulisan ulang kode sumber.
- berorientasi objek.
- memiliki sistem pengelolaan memori otomatis (garbage collection, seperti java)
- modular, mudah dikembangkan dengan menciptakan modul-modul baru; modul-modul tersebut dapat dibangun dengan bahasa Python maupun C/C++.
- memiliki fasilitas pengumpulan sampah otomatis, seperti halnya pada bahasa pemrograman Java, python memiliki fasilitas pengaturan penggunaan ingatan komputer sehingga para pemrogram tidak perlu melakukan pengaturan ingatan komputer secara langsung.
- memiliki banyak fasilitas pendukung sehingga mudah dalam pengoperasiannya.

## BAB II

### ANALISIS PERSOALAN DAN DEKOMPOSISI

#### 2.1 CFG Production

Berikut ini adalah hasil Context-Free Grammar yang telah kami buat terkait membuat compiler bahasa python.

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

#### Non-Terminal Symbol / Variabel (V)

S	IF	INBRACKET	IMPORT	PASS	STRING
VV	ELIF	TYPE	FROM	CONTINUE	INPUT
VAR	ELSE	WHILE	RETURN	COMMENT	CLASS
VAL	PRINT	FOR	RAISE	CONTENT	BREAK
BOOL	EXPRES	RANGE	METHOD		
OPS	RELATION	DEF	RANGE		

#### Terminal Symbol (T)

+	%	number	=	double	class
-	and	is	>	while	import
*	or	!	<	for	as
/	variable	not	(	in	from
true	“	:	)	range	return
false	if	print	int	def	raise
string	elif	input	double	len	range
‘	else	str	float	,	continue
with	open				

**Productions (P)**

<b>Production</b>	<b>Hasil</b>
S	S S   VAR = VV   VAR + = VV   VAR - = VV   VAR * = VV   VAR / = VV   IF   PRINT   WHILE   FOR   DEF   CLASS   IMPORT   FROM   COMMENT   ARRAY;
V	VAR   VAL   VV OPS VV   INPUT   VV , VV   METHOD;
VAR	variable
VAL	number   VV OPS VV   VV * * VV   VV // VV   ( VV )   BOOL   STRING;
RELATION	>   <   = =   ! =   < =   > = ;
BOOL	true   false   BOOL and BOOL   BOOL or BOOL   not BOOL   VV is VV   VV RELATION VV;
OPS	+   -   *   /   %;
STRING	" string "   ' string '   STRING + STRING;
CONTENT	VAR   CONTENT CONTENT;
COMMENT	" " " CONTENT " " "   ' ' ' CONTENT ' ' ';
INBRACKET	( VV )   ( );
PRINT	print INBRACKET;
EXPRES	( BOOL ) : S   BOOL : S;
RAISE	raise INBRACKET
BREAK	break
PASS	pass
CONTINUE	continue;
IF	if EXPRES   IF ELIF   IF ELSE   IF RAISE   IF BREAK   IF PASS   IF CONTINUE;
ELIF	elif EXPRES   ELIF ELIF   ELIF ELSE;
ELSE	else : S;
TYPE	str   float   int   double;
INPUT	input INBRACKET   TYPE ( INPUT );

RANGE	range INBRACKET;
WHILE	while EXPRES;
FOR	for VAR in STRING : S   for VAR in RANGE : S;
RETURN	return BOOL   return VAL;
DEF	def VAR INBRACKET : S   DEF RETURN;
CLASS	class VAR : S;
IMPORT	import VAR as VAR   import VAR;
FROM	from VAR IMPORT;
METHOD	len INBRACKET   with open INBRACKET as VAR

**START SYMBOL (S) : S**

## 2.2 CNF (Chomsky Normal Form)

Berikut ini adalah hasil Context-Free Grammar yang sudah menjadi bentuk Chomsky Normal Form yang telah kami dapatkan dari converter CFG ke CNF.

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

**Non-Terminal Symbol / Variabel (V)**

S	IF	INBRACKET	IMPORT	PASS	STRING
VV	ELIF	TYPE	FROM	CONTINUE	INPUT
VAR	ELSE	WHILE	RETURN	COMMENT	CLASS
VAL	PRINT	FOR	RAISE	CONTENT	BREAK
BOOL	EXPRES	RANGE	METHOD		
OPS	RELATION	DEF	RANGE		

**Terminal Symbol (T)**

+	%	number	=	double	class
-	and	is	>	while	import

*	or	!	<	for	as
/	variable	not	(	in	from
true	“	:	)	range	return
false	if	print	int	def	raise
string	elif	input	double	len	range
‘	else	str	float	,	continue
with	open				

### Productions (P)

Production	Hasil
S	S S   VAR A1   VAR B1   VAR C1   VAR D1   VAR E1   J3 EXPRES   IF ELIF   IF ELSE   IF RAISE   IF BREAK   IF PASS   IF CONTINUE   M3 INBRACKET   E3 EXPRES   D3 A11   D3 B11   A3 C11   DEF RETURN   Z2 D11   Y2 E11   Y2 VAR   W2 F11   O3 S1   N3 T1
S1	N3 T1
Z3	=
A1	Z3 W
B1	OPS B2
B2	Z3 VV
C1	OPS C2
C2	Z3 VV
D1	OPS D2
D2	Z3 VV
E1	OPS E2
E2	Z3 VV
VV	VAR   VV F1   VV G1   number   VV H1   VV I1   VV J1   X3 K1   G3 INBRACKET   TYPE Z1   U2 INBRACKET

	T2 G11   true   false   BOOL L1   BOOL M1   R3 BOOL   VV N1   VV O1   O3 P1   N3 Q1   STRING R1
F1	OPS VV
Y3	,
G1	Y3 VV
VAR	variable
VAL	umber   VV H1   VV I1   VV J1   X3 K1   true   false   BOOL L1   BOOL M1   R3 BOOL   VV N1   VV O1   O3 P1   N3 Q1   STRING R1
H1	OPS VV
I1	OPS I2
I2	OPS VV
J1	OPS J2
J2	OPS VV
X3	(
W3	)   L3 S
K1	VV W3
RELATION	>   <   Z3 Z3   U3 Z3   RELATION Z3   RELATION Z3
U3	!
BOOL	true   false   BOOL L1   BOOL M1   R3 BOOL   VV N1   VV O1
T3	and   CONTENT T4
L1	T3 BOOL
S3	or   CONTENT S4
M1	S3 BOOL
R3	not
Q3	is
N1	Q3 VV
O1	RELATION VV
OPS	+   -   *   /   %

P3	string
O3	“
STRING	O3 P1   N3 Q1   STRING R1
P1	P3 O3
N3	‘
Q1	P3 N3
R1	OPS STRING
CONTENT	CONTENT CONTENT   variable
COMMENT	O3 S1   N3 T1
S1	O3 S2
S2	O3 S3   open
S4	O3 S5
S5	O3 O3
T1	N3 T2
T2	N3 T3   with
T4	N3 T5
T5	N3 N3
INBRACKET	X3 U1   X3 W3
U1	VV W3
M3	print
PRINT	M3 INBRACKET
L3	:
EXPRES	X3 W1   BOOL X1
W1	BOOL W2
W2	W3 W3   from
X1	L3 S
K3	raise
RAISE	K3 INBRACKET
BREAK	break



PASS	pass
CONTINUE	continue
J3	if
IF	J3 EXPRES   IF ELIF   IF ELSE   IF RAISE   IF BREAK   IF PASS   IF CONTINUE
I3	elif
ELIF	I3 EXPRES   ELIF ELIF   ELIF ELSE
H3	else
ELSE	H3 Y1
Y1	L3 S
TYPE	str   float   int   double
G3	input
INPUT	G3 INBRACKET   TYPE Z1
Z1	X3 Z2
Z2	INPUT W3   class
F3	range
RANGE	F3 INBRACKET
E3	while
WHILE	E3 EXPRES
D3	for
C3	in
FOR	D3 A11   D3 B11
A11	VAR A12
A12	C3 A13
A13	STRING A14
A14	L3 S
B11	VAR B12
B12	C3 B13
B13	RANGE B14
B14	L3 S

B3	return
RETURN	B3 BOOL   B3 VAL
A3	def
DEF	A3 C11   DEF RETURN
C11	VAR C12
C12	INBRACKET C13
C13	L3 S
CLASS	Z2 D11
D11	VAR D12
D12	L3 S
Y2	import
X2	as
IMPORT	Y2 E11   Y2 VAR
E11	VAR E12
E12	X2 VAR
FROM	W2 F11
F11	VAR IMPORT
U2	len
METHOD	U2 INBRACKET   T2 G11
G11	S2 G12
G12	INBRACKET G13
G13	X2 VAR
S0	S S   VAR A1   VAR B1   VAR C1   VAR D1   VAR E1   J3 EXPRES   IF ELIF   IF ELSE   IF RAISE   IF BREAK   IF PASS   IF CONTINUE   M3 INBRACKET   E3 EXPRES   D3 A11   D3 B11   A3 C11   DEF RETURN   Z2 D11   Y2 E11   Y2 VAR   W2 F11   O3 S1   N3 T1

**Start Symbol (S) : S**

## BAB III

### IMPLEMENTASI DAN PENGUJIAN

#### SPESIFIKASI TEKNIS PROGRAM

##### 3.1. File CYK2CNF.py

File CYK2CNF.py berisi mengenai prosedur dan fungsi dalam mengkonversi bentuk Context Free-Grammar (CFG) menjadi bentuk Chomsky Normal Form (CNF). Program CYK2CNF.py kami ambil dari referensi <https://github.com/adelmassimo/CFG2CNF>. Program akan mengkonversi CFG yang telah disimpan dalam suatu file .txt yang hasil konversi CNF akan dikeluarkan dalam file out.txt. Berikut ini adalah daftar fungsi / prosedur yang kami gunakan dalam source code CYK2CNF.py.

No.	Fungsi / Prosedur	Tujuan
1	isUnitary	Mengecek apakah suatu rules sudah memiliki productions tepat satu buah symbol terminal dan non-terminal menjadi bagian dari daftar variabel
2	isSimple	Mengecek apakah suatu rules sudah memiliki productions tepat satu buah symbol terminal dan non-terminal menjadi bagian dari daftar variabel V
3	START	Menambahkan suatu start symbol baru bernama S0 ke dalam daftar variabel dan rules yang telah ada
4	TERM	Menghapus rules yang mengandung terminal dan non-terminal simbol sekaligus dan mengubahnya ke bentuk 2 non-terminal atau 1 terminal saja
5	BIN	Prosedur untuk mengeliminasi non-unitry rules

6	DEL	Prosedur untuk melakukan penghapusan non-terminal rules
7	unit_routine	Memeriksa apakah suatu unit atau rules sudah berbentuk unary atau single.
8	UNIT	Mengeliminasi unit production dalam suatu rules
9	convertToMap	Melakukan konversi dari production yang berbentuk list of list menjadi sebuah map.

Tabel 3.1 Daftar fungsi / prosedur dalam converter CFG2CNF.py

### 3.2. File helper.py

File helper.py berisi prosedur dan fungsi untuk melakukan pembersihan terhadap *production* dari CFG sebelum dirubah ke CNF. File helper.py ini merupakan penunjang dari CFG2CNF.py sehingga referensi yang diambil juga berasal dari <https://github.com/adelmassimo/CFG2CNF>. File helper.py ini menunjang proses pembacaan suatu file satu per satu agar bisa dimasukkan dalam list of list yang telah tersedia dalam program. Berikut ini adalah fungsi / prosedur terkait source code helper.py.

No.	Fungsi / Prosedur	Tujuan
1	union	Melakukan penggabungan antara dua 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 lebih lanjut
4	cleanAlphabet	Melakukan pembersihan untuk memuat Terminal dan Variable dalam list
5	seekAndDestroy	Melakukan eliminasi <i>useless</i> variable
6	setupDict	Melakukan penelusuran dari <i>unit</i> variable

7	<code>rewrite</code>	Melakukan pembacaan CFG file format tertentu menjadi format yang bisa dibaca satu per satu dalam array
8	<code>dict2set</code>	Memasukkan dictionary ke dalam suatu list atau set
9	<code>pprintRules</code>	Menampilkan hasil dari rules
10	<code>prettyForm</code>	Melakukan penggabungan untuk production yang memiliki lebih dari 1 hasil

Tabel 3.2 Daftar fungsi / prosedur yang digunakan dalam helper.py

### 3.3. File main.py

File main.py merupakan source code yang berisi main program dari compiler bahasa Python. File ini mengimport file cyk.py yang akan digunakan untuk membership testing terhadap input yang akan dicompile. Secara garis besar, program main.py akan diawali dengan pembacaan CNF dari suatu file yang berisi daftar rules berbentuk CNF. Kemudian program akan membaca input.txt yang berisi input pengguna untuk dijadikan dalam bentuk token-token tertentu. Setelah sudah diolah ke bentuk token, token-token tersebut akan dibaca dalam CYK-Algorithm. Input lolos compile bila akar dari CYK-Table berisi Start Symbol. Berikut ini adalah daftar proses dalam source code main.py

No.	Nama Prose	Penjelasan
1	Load CNF	Mengambil data CNF yang berada dalam file yang berisi rules yang sudah berbentuk CNF.
2.	Tokenize	Setelah mendapatkan input dari pengguna, program tokenizer akan menjadikan input tersebut menjadi token-token yang akan digunakan dalam proses CYK
3.	CYK	Mengecek apakah input yang sudah dijadikan token-token menjadi bagian dari language CFG yang telah didefinisikan

Tabel 3.3 Daftar proses yang digunakan dalam main.py

### 3.4. File CYK.py

File CYK.py berisi prosedur yang terkait dengan Algoritma CYK. Berikut ini adalah isi dari file tersebut.

No.	Fungsi / Prosedur	Fungsi
1	<code>def cyk(tokenizedInput)</code>	Melakukan algoritma CYK untuk suatu input yang sudah di- <i>tokenize</i> , akan mengembalikan tabel CYK itu sendiri

2	<code>def checkValidity(table, wanted)</code>	Melakukan pemeriksaan kepada hasil tabel CYK, apakah setelah CYK dilakukan parsing berhasil dilakukan atau tidak. Akan mengembalikan nilai boolean.
3	<code>LoadCNF (modelPath)</code>	Mengambil data dari CNF yang sudah dibuat untuk dijadikan sebagai map / dictionary dalam CYK Algorithm

Tabel 3.4 Daftar Fungsi atau Prosedur yang digunakan dalam `cyk.py`

### 3.5. File `tokenizeInput.py`

File `CYK.py` berisi prosedur yang terkait dengan Algoritma CYK. Berikut ini adalah isi dari file tersebut.

No.	Fungsi / Prosedur	Fungsi
1	<code>def cyk(tokenizedInput)</code>	Melakukan algoritma CYK untuk suatu input yang sudah di- <i>tokenize</i> , akan mengembalikan tabel CYK itu sendiri

Tabel 3.5 Daftar Fungsi atau Prosedur yang digunakan dalam `tokenizeInput.py`

## BAB IV

### EKSPERIMEN

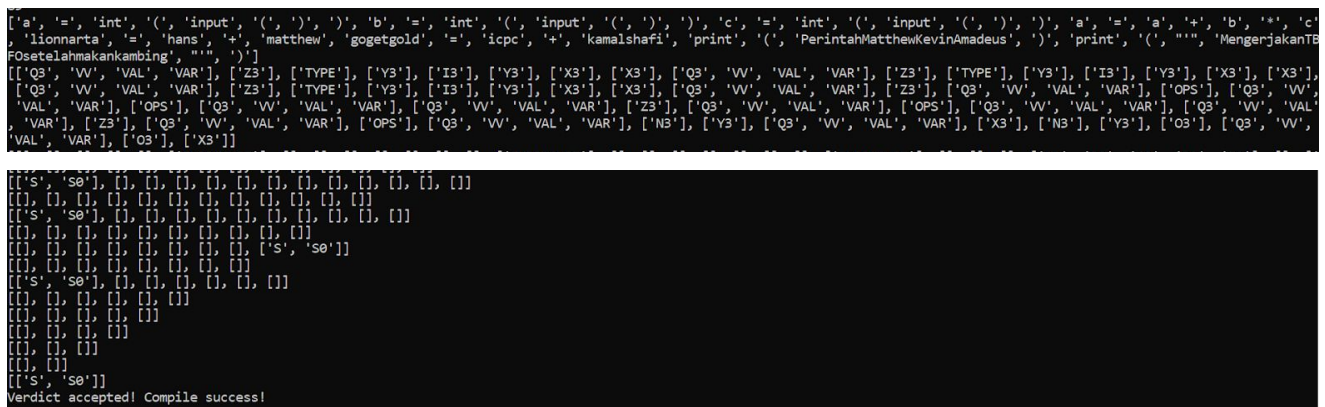
#### Capture Kasus Uji

Dalam bagian ini, kami telah melakukan beberapa kasus uji terhadap compiler bahasa python yang telah kami buat. Berikut ini adalah beberapa kasus uji yang telah kami buat.

#### 4.1 Input dan Output Sederhana

Berikut ini adalah salah satu potongan code berbahasa python sederhana yang akan dicompile dengan menggunakan compiler bahasa python yang telah kami buat.

```
a = int(input())
b = int(input())
c = int(input())
a = a + b * c
lionnarta = hans + matthew
gogetgold = icpc + kamalshafi
print(PerintahMatthewKevinAmadeus)
print('MengerjakanTBFOsetelahmakankambing')
```



```
[ 'a', '=', 'int', '(', 'input', '(', '(', ')', ')', ')', 'b', '=', 'int', '(', 'input', '(', '(', ')', ')', ')', 'c', '=', 'int', '(', 'input', '(', '(', ')', ')', ')', 'a', '=', 'a', '+', 'b', '*', 'c',
, 'lionnarta', '=', 'hans', '+', 'matthew', 'gogetgold', '=', 'icpc', '+', 'kamalshafi', 'print', '(', 'PerintahMatthewKevinAmadeus', ')', 'print', '(', '""', 'MengerjakanTB
FOsetelahmakankambing', '""', ')']
[[ 'Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['TYPE'], ['Y3'], ['I3'], ['Y3'], ['X3'], ['X3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['TYPE'], ['Y3'], ['I3'], ['Y3'], ['X3'], ['X3'],
['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['TYPE'], ['Y3'], ['I3'], ['Y3'], ['X3'], ['X3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['OPS'], ['Q3', 'VV',
, 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['OPS'], ['Q3', 'VV', 'VAL', 'VAR'], ['Q3', 'VV', 'VAL',
, 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['OPS'], ['Q3', 'VV', 'VAL', 'VAR'], ['NB3'], ['Y3'], ['Q3', 'VV', 'VAL', 'VAR'], ['X3'], ['NB3'], ['Y3'], ['Q3'], ['Q3', 'VV',
, 'VAL', 'VAR'], ['Q3'], ['X3']]
[[['s', 'se'], [], [], [], [], [], [], [], [], [], []],
[[], [], [], [], [], [], [], [], [], []],
[['s', 'se'], [], [], [], [], [], [], [], [], []],
[[], [], [], [], [], [], [], [], [], []],
[[], [], [], [], [], [], [], [], ['s', 'se']],
[[], [], [], [], [], [], [], [], []],
[['s', 'se'], [], [], [], [], [], [], []],
[[], [], [], [], [], [], [], []],
[[], [], [], [], [], [], [], []],
[[], [], []],
[[], []],
[['s', 'se']]
Verdict accepted! Compile success!
```

Gambar 4.1 Hasil compile input dan output sederhana

Dari hasil percobaan pertama, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Verdict accepted! Compile success!” yang berarti compile berhasil. Dalam gambar diatas, kami menampilkan hasil CYK kami saat proses parsing dan proses akhir yang menunjukkan start



Pada percobaan pertama ini, grammar yang production yang sering digunakan adalah PRINT, VAL, VAR, VV, STRING, dan OPS. Setiap token yang diperoleh dari hasil lexer akan dicocokkan dengan key-key yang sesuai sehingga memunculkan simbol-simbol non-terminal yang memproduksi string-string tersebut.

Berikut ini adalah salah satu potongan code berbahasa python sederhana yang akan dicompile dengan menggunakan compiler bahasa python yang telah kami buat.

[illegible]

Dari hasil percobaan kedua, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Verdict accepted! Compile success!” yang berarti compile berhasil. Dalam gambar diatas, kami menampilkan hasil CYK kami saat proses parsing dan proses akhir yang menunjukkan start

symbol. Karena start symbol dari input kami adalah  $S_0$ , maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan kedua ini, grammar yang paling dominan digunakan adalah IF, ELIF, ELSE, PRINT, OPS, dan BOOL. Di dalam CFG yang telah kami buat, kami telah mendefinisikan rules-rules tertentu sehingga penggunaan if, elif, dan else bisa saling berkaitan satu sama lain dan mempunyai semantik-semantik tertentu. Dalam hal ini, penggunaan elif harus didahului oleh if di sebelumnya dan penggunaan else harus didahului oleh elif atau if di sebelumnya. BOOL dipakai dalam produksi dari non-terminal IF, ELIF, yaitu suatu kondisi yang dinyatakan oleh non-terminal BOOL.

## 4.3 Looping

Berikut ini adalah potongan kode berbahasa python terkait penggunaan looping baik menggunakan while maupun for looping.

```
while(a < 4):
    g = 2
    a += 1

for i in range(9,3,1):
    print(Nilai)

for i in "asdfg" :
    for j in "abcde" :
        a+=1
```

[illegible][illegible]

*Gambar 4.3 Hasil compile looping*

Dari hasil percobaan kedua, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Verdict accepted! Compile success!” yang berarti compile berhasil. Dalam gambar diatas, kami menampilkan hasil CYK kami saat proses parsing dan proses akhir yang menunjukkan start

symbol. Karena start symbol dari input kami adalah S0, maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan ketiga ini, grammar yang paling dominan digunakan adalah WHILE, FOR, RANGE, BOOL, VAR, VAL, dan PRINT. Seperti pada bagian sebelumnya, hasil input yang terdapat dalam input.txt akan diolah menjadi token-token sehingga dapat dicocokkan nilai tokennya dengan key-key yang tersedia di dalam chomskyGrammar dictionary. Penggunaan while, for, range dapat dikenali dengan pola pembentukan kata per kata yang terangkai seperti tertulis pada CFG yang sudah ditransformasi menjadi bentuk CNF yang kami buat. Untuk penggunaan string, kami belum sepenuhnya mampu mengintegrasikan dengan suatu tipe pembentukan dari class mengingat operator '.' sudah dijadikan split dalam tokenizer yang telah kami buat.

#### 4.4 Class, Import, Def

Berikut ini adalah potongan kode berbahasa python terkait penggunaan class, import, dan def yang akan dicompile dengan compiler bahasa python yang kami buat.

```
import pandas as pd
import matplotlib
from matplotlib import numpy
from matplotlib import numpy as np

def gon(art):
    print('null')

def neraca(bobot):
    for i in "neraca":
        i+=1
    return i

class nullity:
    def nonil():
        rom = 1
    def krypto(a):
        a+=1
    class jir:
        g = 1
```

```
[import, 'pandas', 'as', 'pd', import, 'matplotlib', from, 'matplotlib', import, 'numpy', from, 'matplotlib', import, 'numpy', 'as', 'np', def, 'gon', ('', 'a', 'rt'), '', print, (''', ''', 'null', '', ''), def, 'neraca', ('', 'bobot', 't'), '', for, 'i', 'in', '', 'neraca', '', ':', 'i', 't', '=', '1', return, 'i', ']', class, 'nullity', '', def, 'nonil', ('', 'r', 'rom', '=, '1', def, 'kripto', ('', 'a', 't'), 'g', '+, '=', '1', class, 'jir', '', 'g', '=' '1']
```

```
[['B3'], ['Q3', 'VV', 'VAL', 'VAR'], ['A3'], ['Q3', 'VV', 'VAL', 'VAR'], ['B3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z2'], ['Q3', 'VV', 'VAL', 'VAR'], ['B3'], ['Q3', 'VV', 'VAL', 'VAR'], ['X3'], ['MB'], ['M3'], ['Y3'], ['Q3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Q3'], ['X3'], ['D3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Y3'], ['Q3', 'VV', 'VAL', 'VAR'], ['X3'], ['M3'], ['G3'], ['Q3', 'VV', 'VAL', 'VAR'], ['F3'], ['P3'], ['Q3', 'VV', 'VAL', 'VAR'], ['P3'], ['M3'], ['Q3', 'VV', 'VAL', 'VAR'], ['OPS'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['E3'], ['Q3', 'VV', 'VAL', 'VAR'], ['C3'], ['Q3', 'VV', 'VAL', 'VAR'], ['M3'], ['D3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Y3'], ['X3'], ['M3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['D3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Y3'], ['Q3', 'VV', 'VAL', 'VAR'], ['X3'], ['M3'], ['Q3', 'VV', 'VAL', 'VAR'], ['OPS'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR'], ['C3'], ['Q3', 'VV', 'VAL', 'VAR'], ['M3'], ['Q3', 'VV', 'VAL', 'VAR'], ['Z3'], ['Q3', 'VV', 'VAL', 'VAR']]
```

```
[['S', 'IMPORT', 'S0'], [], ['Z2'], [], ['S', 'IMPORT', 'S0'], ['Z1'], [], [], ['S', 'IMPORT', 'S0'], ['Z1'], [], [], ['S', 'IMPORT', 'S0'], [], ['Z2'], [], [], [], ['J1', 'R1'], [], [], [], ['P1'], [], [], [], [], ['J1', 'R1'], [], [], [], [], ['O1'], [], [], [], ['A1', 'B2', 'C2', 'D2', 'E2'], [], ['RETURN'], [], [], [], [], [], [], ['INBRACKET'], [], [], [], ['A1', 'B2', 'C2', 'D2', 'E2'], [], [], [], [], ['J1', 'R1'], [], [], [], ['A1', 'B2', 'C2', 'D2', 'E2'], [], [], [], []], [], ['A1', 'B2', 'C2', 'D2', 'E2']]
```

*Gambar 4.4 Hasil compile penggunaan class,*

Dari hasil percobaan keempat, hasil yang dikeluarkan sesuai dengan yang diharapkan, yaitu “Verdict accepted! Compile success!” yang berarti compile berhasil. Dalam gambar diatas, kami menampilkan hasil CYK kami saat proses parsing dan proses akhir yang menunjukkan start symbol. Karena start symbol dari input kami adalah S0, maka input diterima dan menjadi bagian dari language CFG Python yang telah kami definisikan sebelumnya.

Pada percobaan keempat ini, grammar CFG yang paling dominan dalam kasus ini adalah import, STRING, DEF, VAR, VAL, VV, dan CLASS. Penggunaan grammar seperti import, def, dan class diperlakukan seperti suatu method sehingga memiliki struktur definisi yang tetap sehingga bila beberapa token dikenali menggunakan key, maka pembacaan string akan menghasilkan simbol non-terminal yang pada akhirnya akan terus membaca sampai pada start symbol. Namun, untuk kasus keempat ini, masih sering terjadi kesalahan atau compile error dalam penggunaannya ketika di dalam def kami menyertakan tipe dari variabel tersebut, seperti def (int a, int b) atau yang lainnya. Untuk fungsi def sendiri masih berlaku pada format def ( STRING\* ).

## 4.5 All In One

Berikut ini adalah potongan code berbahasa python yang mencakup keseluruhan code yang ingin kami coba compile.

```
import pandas as pd
import matplotlib
from matplotlib import numpy
from matplotlib import numpy as
np
def gon(art):
    print('null')
def neraca(bobot):
```







## **BAB V**

### **PENUTUP**

#### **5.1 Kesimpulan**

Dari tugas besar berjudul “Compiler Bahasa Python”, maka program yang kami buat berjalan cukup baik mengingat masih ada banyak kendala atau bug terkait compiler bahasa python yang telah kami buat. Program kami mampu menampung beberapa kasus kecil terkait compiler, namun di kasus yang lebih besar dan lebih asing, program kami masih mengalami beberapa kekurangan dalam melakukan kompilasi terhadap suatu file input.txt

Hal ini mungkin disebabkan oleh grammar CFG berbahasa python yang kami buat belum mampu menampung seluruh test case seperti pada compiler python yang asli.

#### **5.2 Saran**

Berikut ini adalah saran-saran yang bisa kami berikan untuk tugas besar ini.

1. Banyak menggali referensi-referensi terkait Context-Free Grammar.
2. Mempunyai jiwa pembelajar yang kuat dan tangguh dalam mempelajari hal-hal baru.
3. Perbanyak intensitas untuk membuat suatu driver untuk setiap test case yang ada.
4. Lakukan secara step by step atau satu per satu terhadap pengujian program.



## REFERENSI

Fairuzabadi, Muhammad, “Bentuk Normal Chomsky”, diakses pada 24 November 2019 pukul 14.03 WIB, <https://fairuzelsaid.wordpress.com/2011/06/23/bentuk-normal-chomsky/>

Fairuzabadi, Muhammad, “Bentuk Normal Chomsky”, diakses pada 24 November 2019 pukul 11.07 WIB, <https://fairuzelsaid.wordpress.com/2011/06/16/tbo-context-free-grammar-cfg/>

Grahne, G. 2019. Introduction to Theoretical Computer Science

Tim Penulis, “Python”, diakses pada 25 November 2019 pukul 11.20 WIB, [https://id.wikipedia.org/wiki/Python\\_\(bahasa\\_pemrograman\)](https://id.wikipedia.org/wiki/Python_(bahasa_pemrograman))

J.E. Hopcroft, R. Motwani, and J.D. Ullman Introduction to Automata Theory, languages, and Computation, Second-Edition, New York, 2001.

Eisle, Robert, “The CYK Algoritihm”, diakses pada 25 November 2019 pukul 23.59 WIB, <https://www.xarg.org/tools/cyk-algorithm/>

Adelmassimo, “CFG 2 CNF”, diakses pada 25 November 2019 pukul 17.00 WIB, <https://github.com/adelmassimo/CFG2CNF>