

LAPORAN TUGAS BESAR TBFO

“Compiler Bahasa Python”

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan

Mata Kuliah Teori Bahasa Formal dan Otomata (IF2124)

KELAS 02

Dosen :

Dra. Harlili M.Sc. dan Rizal Dwi Prayogo, S.Si., M.Si., M.Sc.



DISUSUN OLEH:

Kelompok 03

“Radio”

Anggota:

Rava Naufal A (13520077)

Dimas Shidqi P (13520087)

Rio Alexander (13520088)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER I TAHUN 2021-2022

Daftar Isi

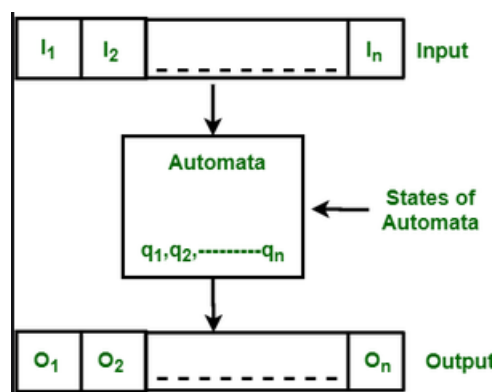
Daftar Isi	2
BAB I Teori Dasar.....	3
1.1. Finite Automata (FA).....	3
1.2. Context-Free Grammar (CFG).....	5
1.3. Bahasa Pemrograman Python.....	8
BAB II Hasil FA dan CFG.....	11
2.1. Hasil FA.....	11
2.2. Hasil CFG.....	12
BAB III Implementasi dan Pengujian.....	23
3.1. Implementasi.....	23
3.2. Pengujian.....	24
BAB IV Kesimpulan dan Saran.....	29
4.1. Kesimpulan.....	29
4.2. Saran.....	29
Referensi	30
Lampiran	31
Lampiran I.....	31
Lampiran II.....	31

BAB I

Teori Dasar

1.1. Finite Automata (FA)

Finite Automata (FA) adalah mesin paling sederhana untuk mengenali suatu pola. *Finite Automata* atau *Finite State Machine* adalah mesin abstrak yang memiliki lima elemen atau tuple. FA memiliki himpunan *states* dan aturan yang berlaku dalam berpindah dari satu *state* ke *state* lainnya, bergantung pada input simbol yang diterima. Pada dasarnya, ini adalah model abstrak dari komputer digital.



Gambar 1

Definisi formal *Finite Automata* sebagai berikut.

$$(Q, \Sigma, q_0, F, \delta)$$

Q : himpunan terbatas dari *state*

Σ : himpunan simbol input

q_0 : *state* awal

F : himpunan *final state*

δ : fungsi transisi

Secara garis besar FA dibagi menjadi dua yaitu, FA dengan *output* dan FA tanpa *output*.

- FA dengan *output* : *Moore Machine* dan *Mealy Machine*
- FA tanpa *output* : DFA, NFA, dan ϵ -NFA

Deterministic Finite Automata (DFA)

DFA didefinisikan dengan 5 tuple $(Q, \Sigma, q_0, F, \delta)$.

Q : himpunan terbatas dari *state*

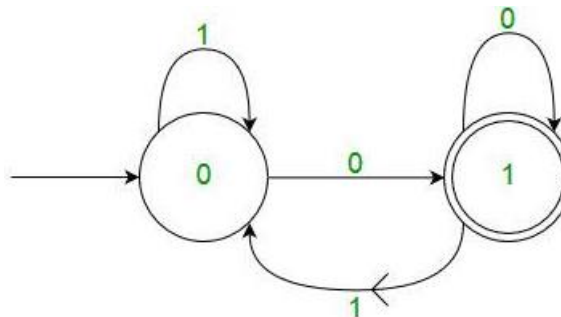
Σ : himpunan simbol input

q_0 : *state* awal

F : himpunan *final state*

δ : fungsi transisi, didefinisikan dengan $\delta (Q \times \Sigma \rightarrow Q)$

Pada DFA, untuk karakter input tertentu, mesin hanya menuju ke satu *state*. Fungsi transisi didefinisikan pada setiap keadaan untuk setiap simbol input. Selain itu, pada DFA *null* (atau ϵ) pemindahan tidak diperbolehkan, yaitu, DFA tidak dapat berpindah *state* tanpa karakter input apa pun. Sebagai contoh, DFA dengan $\Sigma = \{0, 1\}$ menerima semua *string* yang diakhiri dengan '0', sebagai berikut.



Gambar 2

Banyak kemungkinan DFA lain yang dapat dibentuk untuk mendefinisikan suatu pola tertentu, akan tetapi sangat dianjurkan untuk memilih DFA dengan jumlah *state* paling sedikit agar efisien.

Non-Deterministic Finite Automata (NFA)

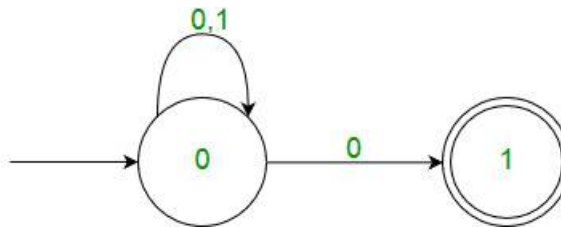
Non-Deterministic Finite Automata (NFA) memiliki kemiripan dengan DFA, akan tetapi pada NFA,

- Perpindahan *null* (atau ϵ) diperbolehkan, yaitu dapat bergerak ke *state* berikutnya tanpa membaca simbol input
- Memiliki kemampuan untuk berpindah ke sejumlah *state* untuk suatu input tertentu

Pada fungsi transisi, DFA dan NFA juga memiliki perbedaan, yaitu fungsi transisi pada NFA didefinisikan sebagai berikut.

$$\delta: Q \times (\Sigma \cup \epsilon) \rightarrow 2^Q$$

Salah satu contoh dari NFA adalah sebagai berikut.



Gambar 3

Pada NFA, jika terdapat jalur untuk *string* input mengarah ke *final state*, maka *string* input diterima. Seperti contoh NFA di atas, terdapat beberapa jalur untuk *string* input “00”, akan tetapi karena salah satu jalur mengarah ke *final state*, maka “00” diterima oleh NFA di atas.

Jika diperhatikan, fungsi transisi dari DFA dan NFA, Q adalah subset dari 2^Q , yang berarti Q adalah bagian dari 2^Q . Sehingga dapat diketahui, bahwa setiap DFA adalah NFA, akan tetapi tidak berlaku untuk sebaliknya, sehingga terdapat cara untuk mengonversi NFA ke DFA, yang pasti terdapat DFA ekuivalen dengan NFA.

1.2. Context-Free Grammar (CFG)

Context Free Grammar (CFG) / Bahasa Bebas Konteks adalah sebuah tata bahasa dimana tidak terdapat pembatasan pada hasil produksinya. CFG juga merupakan 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.

CFG terdiri dari seperangkat aturan tata bahasa yang terbatas dan didefinisikan dengan empat tupel, yaitu

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

N : himpunan terbatas variabel

T : himpunan terbatas terminal

P : himpunan terbatas dari aturan produksi

S : start simbol

Sebagai contoh,

$$G = (\{A\}, \{a, b, c\}, P, A)$$

dengan aturan produksi sebagai berikut.

$$P : A \rightarrow aA, A \rightarrow abc$$

Parse Tree

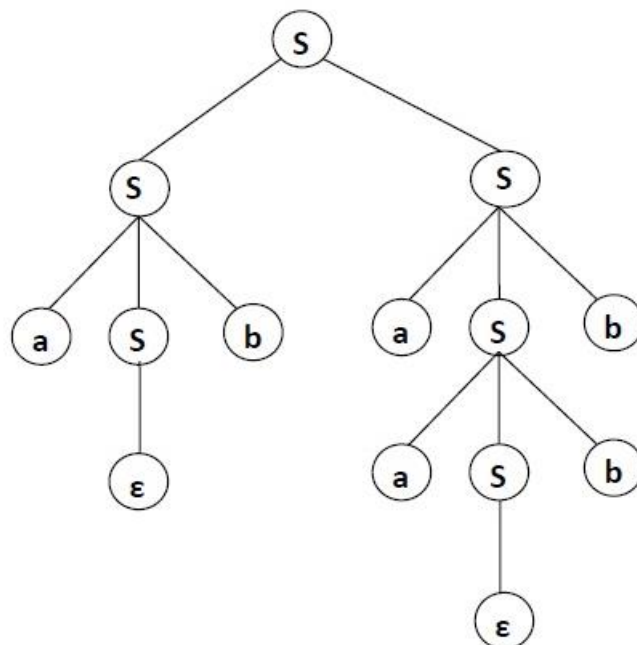
Suatu *derivation tree* atau *parse tree* adalah suatu pohon terurut yang memiliki *root* dan secara grafis mewakili informasi semantik *string* yang berasal dari tata bahasa bebas konteks. *Parse tree* direpresentasikan sebagai berikut.

Root vertex : start simbol

Vertex : non-terminal simbol

Leaves : terminal simbol atau ϵ

Derivation atau hasil pohon dari *parse tree* adalah *string* akhir yang didapatkan dengan menggabungkan seluruh *leaves*/daun pada pohon dari daun terkiri ke daun terkanan. Sebagai contoh, dengan $G = (\{S\}, \{a, b\}, P, S)$, $P : S \rightarrow SS \mid aSb \mid \epsilon$, maka salah satu turunan dari CFG tersebut adalah “abaabb”.



Gambar 4

Proses penurunan/*parsing* dapat dilakukan dengan dua cara, yaitu:

- *Leftmost derivation* : simbol variabel terkiri yang diperluas terlebih dahulu
- *Rightmost derivation*: simbol variabel terkanan yang diperluas terlebih dahulu

Sebagai contoh, bila terdapat aturan produksi $X \rightarrow X+X \mid X*X \mid X \mid a$, dengan alfabet $\{a\}$, maka penurunan untuk *string* “a+a*a” adalah:

- *Leftmost derivation* : $X \rightarrow X+X \rightarrow a+X \rightarrow a+X*X \rightarrow a+a*X \rightarrow a+a*a$
- *Rightmost derivation*: $X \rightarrow X*X \rightarrow X*a \rightarrow X+X*a \rightarrow X+a*a \rightarrow a+a*a$

Left dan Right Recursive Grammars

Dalam tata bahasa bebas konteks G, jika ada produksi dalam bentuk $X \rightarrow Xa$ di mana X adalah non-terminal dan 'a' adalah *string* terminal, hal ini disebut produksi rekursif kiri. Tata bahasa yang memiliki produksi rekursif kiri disebut tata bahasa rekursif kiri (*left recursive grammar*).

Sebaliknya, dalam tata bahasa bebas konteks G, jika ada produksi dalam bentuk $X \rightarrow aX$ di mana X adalah non-terminal dan 'a' adalah *string* terminal, hal ini disebut produksi rekursif kanan. Tata bahasa yang memiliki produksi rekursif kanan disebut tata bahasa rekursif kanan (*right recursive grammar*).

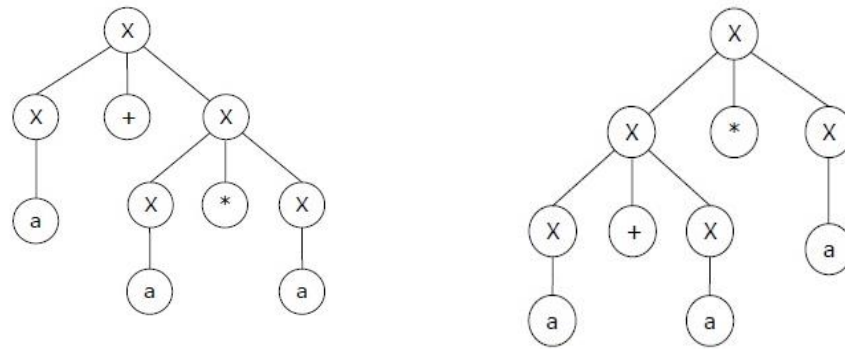
Ambiguitas

Jika tata bahasa bebas konteks G memiliki lebih dari satu pohon turunan untuk beberapa *string* w elemen dari $L(G)$, hal tersebut disebut tata bahasa ambigu. Terdapat beberapa *leftmost derivation* atau *rightmost derivation* untuk beberapa *string* yang dihasilkan dari tata bahasa tersebut. Sebagai contoh, untuk memeriksa apakah suatu tata bahasa G dengan aturan produksi sebagai berikut

$$X \rightarrow X+X \mid X*X \mid X \mid a$$

merupakan suatu tata bahasa ambigu atau tidak.

Untuk memperoleh *string* “a+a*a”, didapatkan dua hasil pohon dengan *leftmost derivation* sebagai berikut.



Gambar 5

Karena terdapat dua *parse trees* untuk string “a+a*a”, maka tata bahasa G termasuk tata bahasa yang ambigu.

1.3. Bahasa Pemrograman Python

Python adalah bahasa pemrograman tingkat tinggi yang serbaguna dan berfokus pada tingkat keterbacaan kode. Python juga mendukung multi paradigma pemrograman, seperti pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Kelebihan lainnya, Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi.

Pada bahasa Python, terdapat 35 kata kunci atau *reserved words*, yang tidak dapat digunakan sebagai *identifier*/pengenal.

Tabel 1. Keyword bahasa pemrogramman python

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

Program Python terdiri dari baris-baris logis dan interpreter mengabaikan baris kosong. Untuk komentar pada Python dapat dilakukan dengan mendeklarasikan karakter '#', sehingga Python akan mengabaikan baris yang memiliki karakter pertama '#'. Hal yang juga menjadi penting pada Python adalah indentasi. Pada kasus seperti membuat percabangan, fungsi, *loop*, atau *class*, diperlukan indentasi pada ekspresi di baris berikutnya.

Python mendukung *single quote* dan *double quote* sebagai string literal, yang harus diperhatikan adalah apabila string diawali dengan *single quote*, maka di akhir string harus menggunakan *single quote* juga. Apabila ditemukan baris kosong pada *source code*, maka Python akan mengabaikan hal tersebut.

Variabel pada Python dapat di-*assign* dengan sebuah nilai dan dapat di *re-assign* atau diubah nilainya seiring waktu, oleh karena itu, Python juga disebut sebagai *dynamically-type language*.

Berikut adalah macam-macam operator yang dapat digunakan pada bahasa pemrograman python.

Tabel 2. Operasi aritmatika pada python

Operasi Aritmatika	
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian desimal
%	Modulus
//	Pembagian bilangan bulat
**	Pangkat

Tabel 3. Operasi logika pada python

Operasi Logika		
not	and	or

Tabel 4. Operasi relasional pada python

Operasi relasional	
<	Lebih kecil dari
>	Lebih besar dari
<=	Lebih kecil sama dengan dari
>=	Lebih besar sama dengan dari
==	Sama dengan
!=	Tidak sama dengan

BAB II

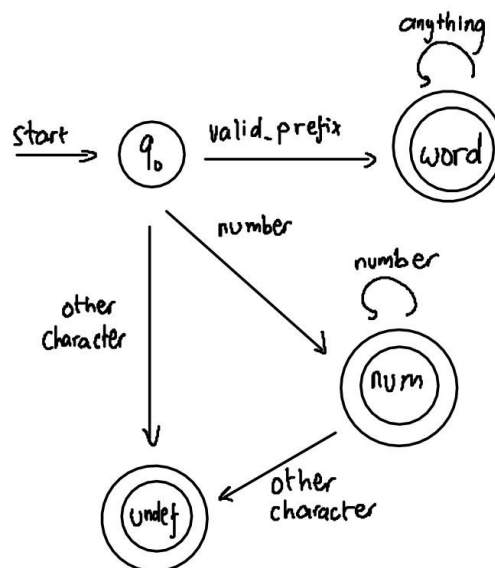
Hasil FA dan CFG

2.1. Hasil FA

Finite Automaton digunakan untuk mengecek apakah sebuah kata merupakan variabel yang valid. Pada finite automaton yang kami buat, kata yang ingin dicek pertama dicocokkan apakah huruf pertama nya merupakan prefix yang valid untuk sebuah variabel, yaitu *underscore* dan *alphabet* baik yang kapital, maupun yang tidak. Apabila valid, maka kata merupakan suatu variabel.

Finite automaton yang kami desain juga dapat mengecek apakah sebuah kata termasuk angka atau bukan variabel dan bukan angka. Apabila semua karakter berupa angka, maka tergolong angka. Dan apabila suatu kata tidak termasuk dalam variabel ataupun angka, maka akan digolongkan sebagai tidak terdefinisi/*undefined*.

Berikut diagram finite automaton yang kami buat.



Gambar 6

2.2. Hasil CFG

Berikut adalah hasil Context-Free Grammar yang telah dibuat.

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

Non-Terminal Symbol / Variabel (V)

BLOCK_CODE	ELIF_STATE_DEF	FOR_LOOP	PARAM_INPUT
IMPORT_FORM	ELIF_STATE_NESTED	ALGORITHM_NESTED	EXPRESSION
IF_STATE	ELIF_ALGORITHM	DEF_STATE	RETURN_STATE
IF_STATE_DEF	ELIF_ALGORITHM_DEF	DEF_ALGORITHM	RETURN_PARAM
IF_STATE_NESTED	ELIF_ALGORITHM_NESTED	DEF_RETURN	CLASS_STATE
IF_ALGORITHM	ELSE_STATE	PARAM_STATE	CLASS_ALGORITHM
IF_ALGORITHM_DEF	ELSE_STATE_DEF	PARAM	SELF_STATE
IF_ALGORITHM_NESTED	ELSE_STATE_NESTED	OPERATOR	DEF_CLASS_STATE
IF_CONDITION	WHILE_LOOP	CONDITION	LIST_FORM
ELIF_STATE	FOR_FORM	PARAM_STATE_INPUT	ELMT_LIST
GET_LIST_IDX	SENTENCE	WRITE_ALGORITHM	ALPHABET
ASSIGN_LIST_IDX	ASSIGNMENT	FLOAT	NUM
STRING	ASSIGN_INTEGER	NEGATIVE_FLOAT	BOOL
STRING_OF_ALPHA	WITH_STATEMENT	NEGATIVE_NUM	IS_FORM
ARITHMETIC_OPERATOR	PRINT_STATE	OPEN_CALL	ERROR
LOGIC_OPERATOR	FUNC_FORM	COMMENT	ASSIGN_DICT
RELATION_OPERATOR	PARAM_STATE_FUNC	QUOTATION_COMMENT_SENTENCE	DICT_CONTENT
ASSIGN_OPERATOR	PARAM_FUNC	RAISE_STATE	COMMENT_QUOTATION

Terminal Symbol (T)

import	case	range	append
from	return	print	and
as	continue	for	or
True	break	open	not
False	pass	write	if
def	raise	num	else
None	in	self	elif
with	class	len	while
word	+	%	:
const	-	=	,
input	*	>	.
is	/	<	#
[]	()
{	}	`	''
'''	!	IOERROR	ValueError
ZeroDivisionError	ImportError	NameError	TypeError

Production (P)

BLOCK_CODE	BLOCK_CODE BLOCK_CODE ASSIGNMENT ASSIGN_INTEGER IMPORT_FORM IF_STATE IF_STATE ELSE_STATE IF_STATE ELIF_STATE WHILE_LOOP FOR_FORM DEF_STATE WITH_STATEMENT WRITE_ALGORITHM PASS ALPHABET POINT APPEND OP_BRACK ALPHABET CS_BRACK OPERATOR PRINT_STATE CLASS_STATE ALPHABET POINT ALPHABET PARAM_STATE FUNC_FORM ASSIGN_DICT OPEN_CALL ASSIGN_LIST_IDX IS_FORM
IMPORT_FORM	IMPORT ALPHABET IMPORT ALPHABET AS ALPHABET FROM ALPHABET IMPORT ALPHABET FROM

	ALPHABET IMPORT ALPHABET AS ALPHABET
IF_STATE	IF_ALGORITHM IF_ALGORITHM ELSE_STATE IF_ALGORITHM ELIF_STATE
IF_STATE_DEF	IF_ALGORITHM_DEF IF_ALGORITHM_DEF ELSE_STATE_DEF IF_ALGORITHM_DEF ELIF_STATE_DEF
IF_STATE_NESTED	IF_ALGORITHM_NESTED IF_ALGORITHM_NESTED ELSE_STATE_NESTED IF_ALGORITHM_NESTED ELIF_STATE_NESTED
IF_ALGORITHM	IF CONDITION COLON BLOCK_CODE IF CONDITION COLON RAISE_STATE
IF_ALGORITHM_DEF	IF CONDITION COLON DEF_ALGORITHM IF CONDITION COLON DEF_RETURN IF CONDITION COLON RAISE_STATE
IF_ALGORITHM_NESTED	IF CONDITION COLON ALGORITHM_NESTED IF CONDITION COLON RAISE_STATE
IF_CONDITION	IF CONDITION
ELIF_STATE	ELIF_ALGORITHM ELIF_ALGORITHM ELSE_STATE ELIF_ALGORITHM ELIF_STATE
ELIF_STATE_DEF	ELIF_ALGORITHM_DEF ELIF_ALGORITHM_DEF ELSE_STATE_DEF ELIF_ALGORITHM_DEF ELIF_STATE_DEF
ELIF_STATE_NESTED	ELIF_ALGORITHM_NESTED ELIF_ALGORITHM_NESTED ELSE_STATE_NESTED ELIF_ALGORITHM_NESTED ELIF_STATE_NESTED
ELIF_ALGORITHM	ELIF CONDITION COLON BLOCK_CODE ELIF CONDITION COLON RAISE_STATE
ELIF_ALGORITHM_DEF	ELIF CONDITION COLON DEF_ALGORITHM ELIF CONDITION

	COLON DEF_RETURN ELIF CONDITION COLON RAISE_STATE
ELIF_ALGORITHM_NESTED	ELIF CONDITION COLON ALGORITHM_NESTED ELIF CONDITION COLON RAISE_STATE
ELSE_STATE	ELSE COLON BLOCK_CODE ELSE COLON RAISE_STATE ELSE IF_STATE
ELSE_STATE_DEF	ELSE COLON DEF_ALGORITHM ELSE COLON DEF_RETURN ELSE IF_STATE_DEF ELSE COLON RAISE_STATE
ELSE_STATE_NESTED	ELSE COLON ALGORITHM_NESTED ELSE IF_STATE_NESTED ELSE COLON RAISE_STATE
WHILE_LOOP	WHILE CONDITION COLON ALGORITHM_NESTED WHILE BOOL COLON ALGORITHM_NESTED WHILE OP_BRACK BOOL CS_BRACK COLON ALGORITHM_NESTED
FOR_FORM	FOR FOR_LOOP COLON ALGORITHM_NESTED
FOR_LOOP	ALPHABET IN ALPHABET ALPHABET IN RANGE OP_BRACK EXPRESSION CS_BRACK ALPHABET IN RANGE OP_BRACK EXPRESSION COMMA EXPRESSION CS_BRACK ALPHABET IN RANGE OP_BRACK EXPRESSION COMMA EXPRESSION COMMA EXPRESSION CS_BRACK
ALGORITHM_NESTED	ALGORITHM_NESTED ALGORITHM_NESTED BREAK CONTINUE ASSIGNMENT ASSIGN_INTEGER IF_STATE_NESTED IF_STATE_NESTED ELSE_STATE_NESTED IF_STATE_NESTED ELIF_STATE_NESTED WHILE_LOOP FOR_FORM WITH_STATEMENT WRITE_ALGORITHM PASS ALPHABET POINT APPEND OP_BRACK ALPHABET CS_BRACK OPERATOR PRINT_STATE FUNC_FORM ASSIGN_LIST_IDX

DEF_STATE	DEF ALPHABET PARAM_STATE COLON DEF_ALGORITHM DEF ALPHABET PARAM_STATE COLON DEF_RETURN DEF ALPHABET PARAM_STATE COLON PRINT_STATE
DEF_ALGORITHM	DEF_ALGORITHM DEF_ALGORITHM ASSIGNMENT ASSIGN_INTEGER IF_STATE_DEF IF_STATE_DEF ELSE_STATE_DEF IF_STATE_DEF ELIF_STATE_DEF WHILE_LOOP FOR_FORM DEF_STATE WITH_STATEMENT WRITE_ALGORITHM PASS ALPHABET POINT APPEND OP_BRACK ALPHABET CS_BRACK OPERATOR PRINT_STATE CLASS_STATE IF CONDITION COLON DEF_RETURN SELF_STATE ALGORITHM_NESTED DEF_RETURN
DEF_RETURN	DEF_ALGORITHM RETURN_STATE RETURN_STATE
PARAM_STATE	OP_BRACK PARAM CS_BRACK OP_BRACK CS_BRACK
PARAM	ALPHABET ALPHABET COMMA PARAM ALPHABET EQUAL STRING COMMA PARAM ALPHABET EQUAL NUM ALPHABET EQUAL STRING ALPHABET EQUAL ALPHABET COMMA PARAM ALPHABET EQUAL ALPHABET FUNC_FORM SELF COMMA PARAM SELF ALPHABET EQUAL DICT
OPERATOR	INPUT OP_BRACK ALPHABET CS_BRACK OUTPUT OP_BRACK ALPHABET CS_BRACK ALPHABET EQUAL EXPRESSION
CONDITION	OP_BRACK CONDITION CS_BRACK CONDITION LOGIC_OPERATOR CONDITION EXPRESSION RELATION_OPERATOR EXPRESSION EXPRESSION RELATION_OPERATOR LIST_FORM EXPRESSION IN EXPRESSION NOT CONDITION ALPHABET BOOL FUNC_FORM SELF_STATE RELATION_OPERATOR EXPRESSION

PARAM_STATE_INPUT	OP_BRACK PARAM_INPUT CS_BRACK OP_BRACK CS_BRACK
PARAM_INPUT	ALPHABET STRING NUM ALPHABET COMMA PARAM_INPUT STRING COMMA PARAM_INPUT NUM COMMA PARAM_INPUT
EXPRESSION	FLOAT ALPHABET NUM NEGATIVE_NUM FUNC_FORM NEGATIVE_FLOAT OP_BRACK EXPRESSION CS_BRACK EXPRESSION ARITHMATIC_OPERATOR EXPRESSION LEN OP_BRACK ALPHABET CS_BRACK LEN OP_BRACK ALPHABET CS_BRACK EXPRESSION ALPHABET POINT ALPHABET NONE STRING ALPHABET OP_SQ_BRACK INDEX CS_SQ_BRACK BOOL
INDEX	ALPHABET NUM NEGATIVE_NUM INDEX ARITHMATIC_OPERATOR INDEX LEN OP_BRACK ALPHABET CS_BRACK LEN OP_BRACK ALPHABET CS_BRACK INDEX ALPHABET POINT ALPHABET ALPHABET OP_SQ_BRACK INDEX CS_SQ_BRACK
RETURN_STATE	RETURN RETURN_PARAM
RETURN_PARAM	ALPHABET ALPHABET COMMA RETURN_PARAM NUM NEGATIVE_NUM FLOAT NEGATIVE_FLOAT STRING BOOL
CLASS_STATE	CLASS ALPHABET COLON CLASS_ALGORITHM
CLASS_ALGORITHM	CLASS_ALGORITHM CLASS_ALGORITHM ASSIGNMENT ASSIGN_INTEGER IMPORT_FORM IF_ALGORITHM IF_ALGORITHM ELSE_STATE IF_ALGORITHM ELIF_STATE WHILE_LOOP FOR_FORM DEF_CLASS_STATE WITH_STATEMENT DEF_STATE WRITE_ALGORITHM PASS ALPHABET POINT APPEND OP_BRACK ALPHABET CS_BRACK OPERATOR PRINT_STATE
SELF_STATE	SELF POINT ALPHABET

DEF_CLASS_STATE	DEF ALPHABET OP_BRACK SELF COMMA PARAM CS_BRACK BLOCK_CODE DEF ALPHABET OP_BRACK SELF COMMA PARAM CS_BRACK DEF_RETURN
LIST_FORM	OP_SQ_BRACK CS_SQ_BRACK OP_SQ_BRACK ELMT_LIST CS_SQ_BRACK OP_SQ_BRACK LIST_FORM CS_SQ_BRACK OP_SQ_BRACK ALPHABET FOR FOR_LOOP CS_SQ_BRACK
ELMT_LIST	LIST_FORM COMMA LIST_FORM LIST_FORM COMMA ELMT_LIST ELMT_LIST COMMA ELMT_LIST ELMT_LIST COMMA LIST_FORM EXPRESSION EXPRESSION COMMA LIST_FORM NUM NUM COMMA LIST_FORM STRING STRING COMMA LIST_FORM
GET_LIST_IDX	ALPHABET OP_SQ_BRACK ALPHABET CS_SQ_BRACK ALPHABET OP_SQ_BRACK NUM CS_SQ_BRACK GET_LIST_IDX OP_SQ_BRACK ALPHABET CS_SQ_BRACK GET_LIST_IDX OP_SQ_BRACK NUM CS_SQ_BRACK
ASSIGN_LIST_IDX	GET_LIST_IDX ASSIGN_OPERATOR EXPRESSION
STRING	QUOTE_MARK STRING_OF_ALPHA QUOTE_MARK DQUOTE_MARK STRING_OF_ALPHA DQUOTE_MARK QUOTE_MARK QUOTE_MARK DQUOTE_MARK DQUOTE_MARK
STRING_OF_ALPHA	STRING_OF_ALPHA STRING_OF_ALPHA ALPHABET ALPHABET STRING_OF_ALPHA
SENTENCE	ALPHABET ALPHABET SENTENCE
ASSIGNMENT	ALPHABET EQUAL EXPRESSION ALPHABET EQUAL STRING ALPHABET EQUAL LIST_FORM ALPHABET EQUAL FUNC_FORM ALPHABET EQUAL INPUT PARAM_STATE_INPUT SELF_STATE EQUAL EXPRESSION SELF_STATE ASSIGN_OPERATOR EXPRESSION

ASSIGN_INTEGER	ALPHABET ASSIGN_OPERATOR EXPRESSION
WITH_STATEMENT	WITH OPEN_CALL AS ALPHABET COLON BLOCK_CODE
WRITE_ALGORITHM	ALPHABET POINT WRITE OP_BRACK STRING CS_BRACK ALPHABET POINT WRITE OP_BRACK EXPRESSION CS_BRACK ALPHABET POINT WRITE OP_BRACK ALPHABET CS_BRACK
FLOAT	NUM POINT NUM
NEGATIVE_FLOAT	MINUS FLOAT
NEGATIVE_NUM	MINUS NUM
ALPHABET	word
NUM	num
BOOL	True False
IS_FORM	EXPRESSION IS EXPRESSION ALPHABET IS GET_LIST_IDX ALPHABET IS LIST_FORM
ARITHMATIC_OPERATOR	MINUS PLUS MUL DIV MOD MUL MUL
LOGIC_OPERATOR	AND OR NOT
RELATION_OPERATOR	GREATER LESS GREATER EQUAL LESS EQUAL EQUAL EQUAL EXC_MARK EQUAL IS_FORM
ASSIGN_OPERATOR	EQUAL PLUS EQUAL MINUS EQUAL MUL EQUAL DIV EQUAL
PRINT_STATE	PRINT OP_BRACK CONDITION CS_BRACK PRINT OP_BRACK STRING CS_BRACK PRINT OP_BRACK ALPHABET CS_BRACK PRINT OP_BRACK EXPRESSION CS_BRACK PRINT OP_BRACK FUNC_FORM CS_BRACK PRINT OP_BRACK CS_BRACK PRINT OP_BRACK IS_FORM CS_BRACK
FUNC_FORM	ALPHABET PARAM_STATE_FUNC ALPHABET POINT FUNC_FORM ALPHABET POINT OPEN_CALL

	ALPHABET OP_BRACK STRING CS_BRACK
PARAM_STATE_FUNC	OP_BRACK PARAM_FUNC CS_BRACK OP_BRACK CS_BRACK
PARAM_FUNC	ALPHABET STRING NUM ALPHABET COMMA PARAM_FUNC STRING COMMA PARAM_FUNC NUM COMMA PARAM_FUNC ALPHABET EQUAL PARAM_FUNC SELF FUNC_FORM ALPHABET POINT FUNC_FORM
OPEN_CALL	OPEN PARAM_STATE
COMMENT	HASHTAG SENTENCE COMMENT_QUOTATION QUOTATION_COMMENT_SENTENCE COMMENT_QUOTATION
QUOTATION_COMMENT_SENTENCE	SENTENCE SENTENCE QUOTATION_COMMENT_SENTENCE
RAISE_STATE	RAISE ERROR OP_BRACK STRING CS_BRACK
ERROR	ZeroDivisionError ValueError TypeError IOError NameError ImportError
ASSIGN_DICT	ALPHABET EQUAL OP_CURLY_BRACK CS_CURLY_BRACK ALPHABET EQUAL OP_CURLY_BRACK DICT_CONTENT CS_CURLY_BRACK
DICT_CONTENT	EXPRESSION COLON EXPRESSION DICT_CONTENT COMMA DICT_CONTENT
QUOTE_MARK	'
COMMENT_QUOTATION	QUOTE_MARK QUOTE_MARK QUOTE_MARK
POINT	.
IF	if
ELIF	elif
ELSE	else
IMPORT	import

FROM	from
AS	as
DEF	def
NONE	None
WITH	with
RETURN	return
CONTINUE	continue
BREAK	break
PASS	pass
RAISE	raise
IN	in
CLASS	class
MINUS	-
PLUS	+
MUL	*
DIV	/
MOD	%
AND	and
OR	or
NOT	not
EQUAL	=
WHILE	while
FOR	for
INPUT	input
COLON	:
COMMA	,
OP_SQ_BRACK	[

CS_SQ_BRACK]
OP_BRACK	(
CS_BRACK)
OP_CURLY_BRACK	{
CS_CURLY_BRACK	}
GREATER	>
LESS	<
DQUOTE_MARK	"
HASHTAG	#
RANGE	range
PRINT	print
EXC_MARK	!
OPEN	open
SELF	self
APPEND	append
LEN	len
IS	is
WRITE	write

Start Symbol (S) : BLOCK_CODE

BAB III

Implementasi dan Pengujian

3.1. Implementasi

CFG2CNF.py

<code>def read_grammar(filename)</code>	Mengubah grammar dari file txt menjadi list berisi production rule tiap baris
<code>def convert_large_rules(grammar)</code>	Memecah production rule yang memiliki lebih dari 2 variabel menjadi maksimal 2 variabel
<code>def convert_unit_productions(grammar, terminal, terminal_rule)</code>	Mengubah grammar menjadi unit production terkecil
<code>def search_rule(grammar, rule_nonterm)</code>	Mengembalikan index sebuah <i>rule</i> dengan variabel yang dicari dalam grammar
<code>def write_to_file(grammar)</code>	Menyimpan grammar yang telah diubah menjadi file txt
<code>def convert_grammar(filename)</code>	Prosedur mengubah CFG menjadi CNF menggunakan fungsi diatas

tokenizer.py

<code>def varname_checker(word)</code>	Mendeteksi apakah token yang bukan <i>keyword</i> berbentuk <i>word/variabel</i> yang valid, <i>number</i> , atau <i>undefined</i> menggunakan finite automata
<code>def tokenize_file(filePath)</code>	Membaca file menjadi token dan mengubah token tersebut menjadi bentuk terminal

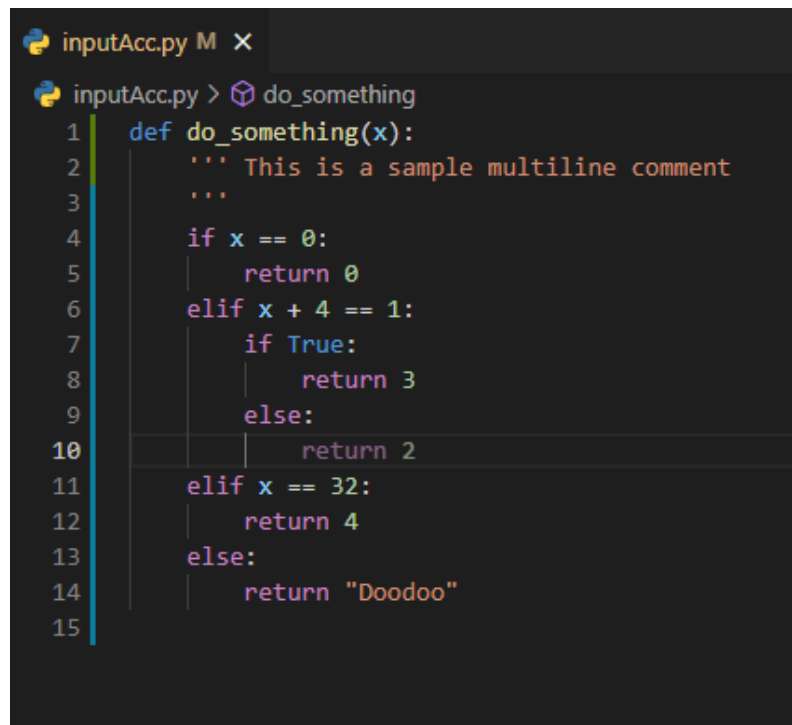
cyk.py

<code>def cyk_algorithm(grammar, tokenized_input)</code>	Mengaplikasikan algoritma CYK pada token yang sudah diolah pada file tokenizer.py untuk mengecek apakah <i>syntax error</i> atau tidak
<code>def line_error_checker(check_table, tokenized_lines, isAccepted)</code>	Mengecek baris-baris pada file uji coba yang memiliki kemungkinan <i>error</i> pada saat kompilasi

3.2. Pengujian

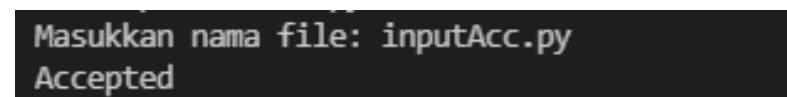
Uji Coba 1

Pada uji coba pertama dilakukan dengan file `inputAcc.py` yang terdapat pada spesifikasi tugas besar, dan dihasilkan output “Accepted” karena program lulus *compile*. File uji coba dan hasil *output* dapat dilihat sebagai berikut.



```
inputAcc.py M X
inputAcc.py > do_something
1 def do_something(x):
2     ''' This is a sample multiline comment
3     '''
4     if x == 0:
5         return 0
6     elif x + 4 == 1:
7         if True:
8             return 3
9         else:
10            return 2
11     elif x == 32:
12         return 4
13     else:
14         return "Doodoo"
15
```

Gambar 7

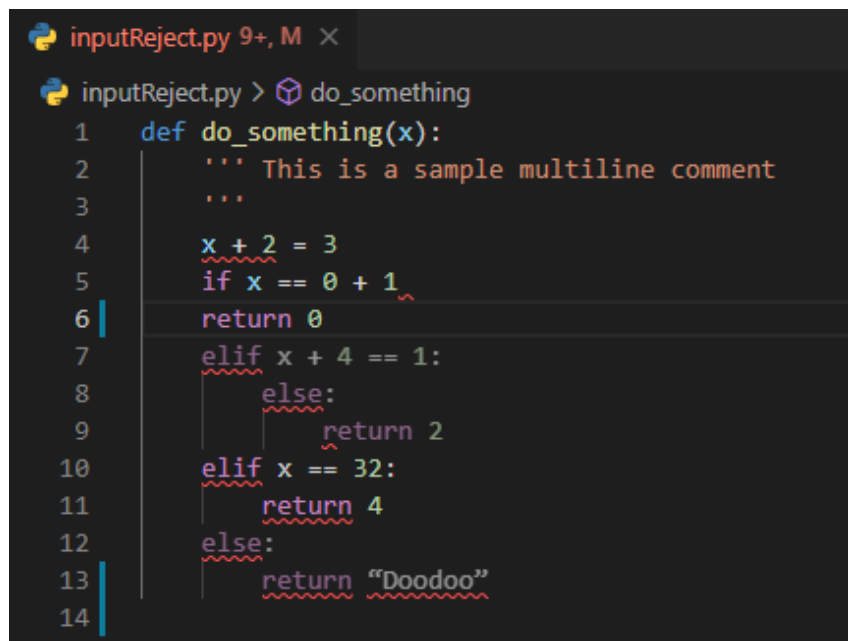


```
Masukkan nama file: inputAcc.py
Accepted
```

Gambar 8

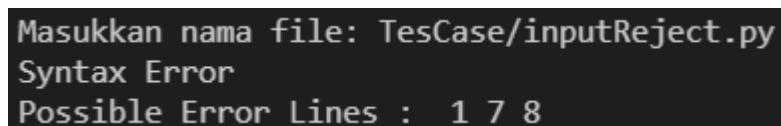
Uji Coba 2

Pada uji coba yang kedua dengan file `inputReject.py` seperti yang terdapat pada spesifikasi tugas besar, didapatkan hasil *output* “Syntax Error”. Hal ini dikarenakan terdapat kode seperti “`x + 2 = 3`”, *if conditional* yang tidak memakai tanda titik dua, dan beberapa *possible error lines* yang pada program dicatat sebagai “*parent*” dari kemungkinan *error* yang ada. File uji coba dan hasil *output* dapat dilihat sebagai berikut.



```
inputReject.py 9+, M x
inputReject.py > do_something
1 def do_something(x):
2     ''' This is a sample multiline comment
3     ...
4     x + 2 = 3
5     if x == 0 + 1
6     return 0
7     elif x + 4 == 1:
8         else:
9             return 2
10    elif x == 32:
11        return 4
12    else:
13        return "Doodoo"
14
```

Gambar 9




```
Masukkan nama file: TesCase/inputReject.py
Syntax Error
Possible Error Lines : 1 7 8
```

Gambar 10

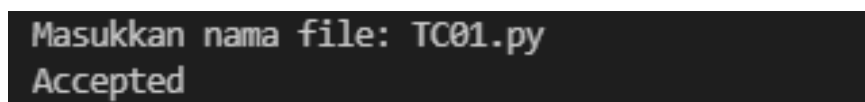
Uji Coba 3

Uji coba ketiga dilakukan dengan file `TC01.py` dan dihasilkan *output* “Accepted”, karena secara sintaks program valid. File uji coba dan hasil *output* dapat dilihat sebagai berikut.



```
TC01.py 2 x
TesCase > TC01.py > ...
1  from flask import Flask
2  import json
3  import random as rn
4
5
6  app = Flask(__name__)
7
8  def payment():
9      VIRTUAL_ACCOUNT_LENGTH = 10
10
11      data = request.get_json()
12      print(data)
13      shopping_cart_id = data[1]
14
15      rn.seed(shopping_cart_id)
16
17      # virtual_account = '7771674623'
18      virtual_account = ''
19      for _ in range(VIRTUAL_ACCOUNT_LENGTH):
20          rand = rn.randint(1, 9)
21          virtual_account += str(rand)
22
23      total_price = get_shopping_cart_total(shopping_cart_id)
24
25      payload = json.dumps({
26          'VIRTUAL_ACCOUNT': virtual_account,
27          'TOTAL_PRICE': total_price
28      })
29
30      return payload
31  app.add_url_rule('/payment', 'payment', payment)
32
33  def get_shopping_cart_total(shopping_cart_id = 3):
34      total = rn.randint(5, 2000) * 1000 - rn.randint(0, 999)
35      return total
36
37  if __name__ == '__main__':
38      app.run()
39
```

Gambar 11

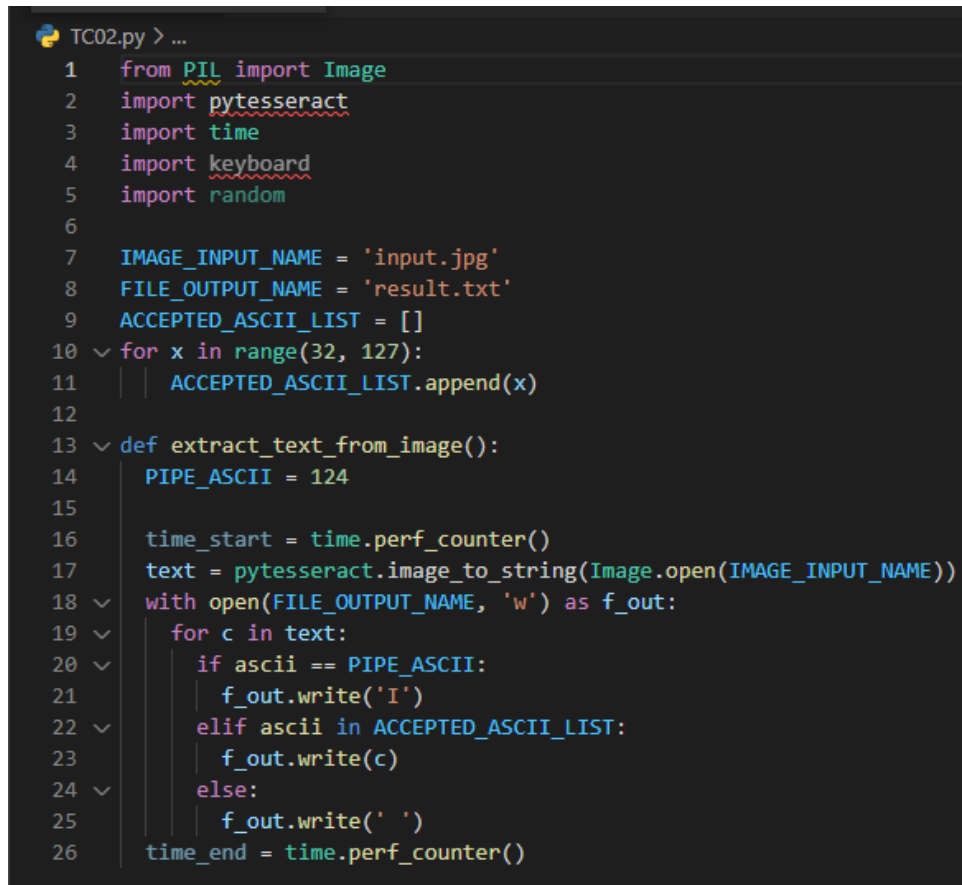


```
Masukkan nama file: TC01.py
Accepted
```

Gambar 12

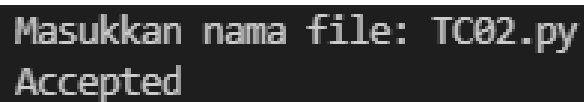
Uji Coba 4

Uji coba keempat dilakukan dengan file `TC02.py` dan dihasilkan *output* “Accepted”, karena secara sintaks program dapat dinyatakan valid. File uji coba dan hasil *output* dapat dilihat sebagai berikut.



```
TC02.py > ...
1  from PIL import Image
2  import pytesseract
3  import time
4  import keyboard
5  import random
6
7  IMAGE_INPUT_NAME = 'input.jpg'
8  FILE_OUTPUT_NAME = 'result.txt'
9  ACCEPTED_ASCII_LIST = []
10 for x in range(32, 127):
11     ACCEPTED_ASCII_LIST.append(x)
12
13 def extract_text_from_image():
14     PIPE_ASCII = 124
15
16     time_start = time.perf_counter()
17     text = pytesseract.image_to_string(Image.open(IMAGE_INPUT_NAME))
18     with open(FILE_OUTPUT_NAME, 'w') as f_out:
19         for c in text:
20             if ascii == PIPE_ASCII:
21                 f_out.write('I')
22             elif ascii in ACCEPTED_ASCII_LIST:
23                 f_out.write(c)
24             else:
25                 f_out.write(' ')
26     time_end = time.perf_counter()
```

Gambar 13

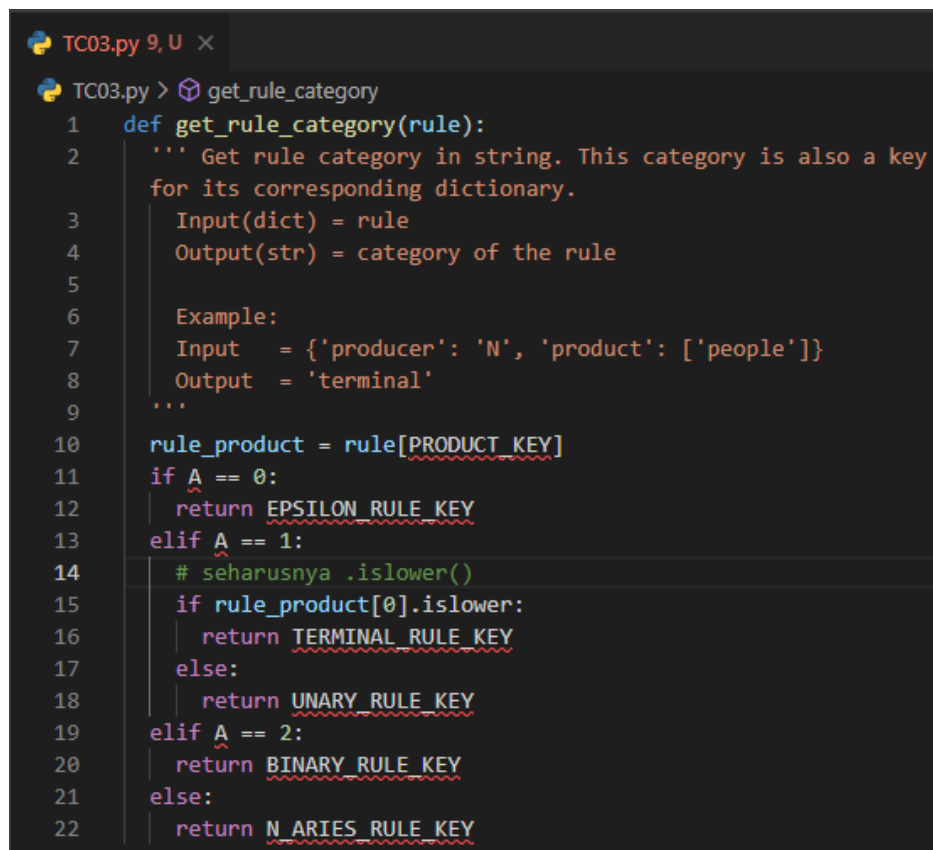


```
Masukkan nama file: TC02.py
Accepted
```

Gambar 14

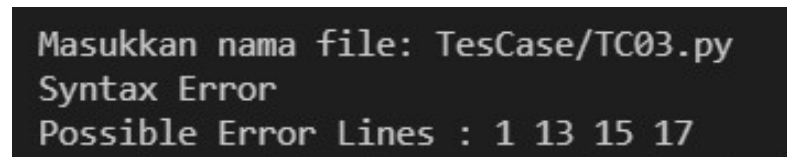
Uji Coba 5

Pada uji coba kelima dilakukan dengan file `TC03.py` dan program menghasilkan *output* “Syntax Error”. Hal ini dikarenakan terdapat kesalahan pada fungsi `islower`, yang seharusnya dinyatakan dengan menggunakan tanda kurung menjadi `islower()`, dan beberapa *possible error lines* yang pada program dicatat sebagai “parent” dari kemungkinan *error* yang ada. File uji coba dan hasil *output* program dapat dilihat sebagai berikut.



```
TC03.py 9, U x
TC03.py > get_rule_category
1 def get_rule_category(rule):
2     ''' Get rule category in string. This category is also a key
3       for its corresponding dictionary.
4       Input(dict) = rule
5       Output(str) = category of the rule
6
7       Example:
8       Input  = {'producer': 'N', 'product': ['people']}
9       Output = 'terminal'
10    ...
11    rule_product = rule[PRODUCT_KEY]
12    if A == 0:
13        return EPSILON_RULE_KEY
14    elif A == 1:
15        # seharusnya .islower()
16        if rule_product[0].islower:
17            return TERMINAL_RULE_KEY
18        else:
19            return UNARY_RULE_KEY
20    elif A == 2:
21        return BINARY_RULE_KEY
22    else:
23        return N_ARIES_RULE_KEY
```

Gambar 15



```
Masukkan nama file: TesCase/TC03.py
Syntax Error
Possible Error Lines : 1 13 15 17
```

Gambar 16

BAB IV

Kesimpulan dan Saran

4.1. Kesimpulan

Dari tugas besar Mata Kuliah Teori Bahasa Formal dan Automata (IF2124) yang berjudul “Compiler Bahasa Python”, program yang telah kami buat berjalan cukup baik, walaupun masih terdapat kendala terkait hasil program ini. Pada kasus dengan tingkat sederhana hingga menengah, program kami dapat melakukan kompilasi dari file .txt yang dibaca dengan cukup baik. Akan tetapi, pada tingkatan kasus yang lebih tinggi, masih ditemukan adanya kesalahan kompilasi.

Hal ini dikarenakan CFG yang kami buat belum sepenuhnya menangani semua kasus yang terdapat pada bahasa pemrograman python dan belum mampu menyerupai *compiler* python yang sesungguhnya.

4.2. Saran

Saran-saran yang dapat kami berikan pada tugas besar ini antara lain, perbanyak mencari referensi terkait teori Context-Free Grammar, membuat struktur CFG yang rapih, mencoba *test case* yang variatif, selalu berusaha apabila menemukan *error* pada program, menjaga komunikasi antar anggota kelompok, dan *debugging* bersama apabila ada kendala, karena hal tersebut dapat melengkapi kesalahan satu sama lain.

Referensi

[Python Syntax \(w3schools.com\)](https://www.w3schools.com/python/python_syntax.asp)

[Python - Basic Syntax \(tutorialspoint.com\)](https://www.tutorialspoint.com/python/python_syntax.htm)

[A simple interpreter from scratch in Python \(part 1\) — jayconrod.com](https://jayconrod.com/posts/47-a-simple-interpreter-from-scratch-in-python-part-1/)

[Context-Free Grammar Introduction \(tutorialspoint.com\)](https://www.tutorialspoint.com/context-free-grammar/context-free-grammar-introduction.htm)

[Introduction of Finite Automata - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-of-finite-automata/)

Lampiran

Lampiran I

Link Repository Github : [Audino723/Tubes-TBFO-Python-CYK-Compiler \(github.com\)](https://github.com/Audino723/Tubes-TBFO-Python-CYK-Compiler)

Lampiran II

Tabel 5. Pembagian Tugas

No.	Tugas	Penanggung Jawab
1	Context-Free Grammar	Rava N A/13520077
2	Tokenize file	Dimas S P/13520087
3	Cocke-Younger-Kasami	Rio A A/13520088