

Tugas Besar
IF2124
Teori Bahasa Formal dan
Automata
COMPILER BAHASA PYTHON



Disiapkan oleh:

Kelas 01 / Kelompok No Context Language

Dzaky Fattan Rizqullah	13520004
Shadiq Harwiz	13520038
Farrel Farandieka Fibriyanto	13520054

Teknik Informatika
Sekolah Teknik Elektro dan
Informatika Institut Teknologi
Bandung
2021

DASAR TEORI

A. *Python*

Python adalah bahasa pemrograman yang interaktif dan berorientasi objek, diciptakan oleh Guido van Rossum pada tahun 1990. Bahasa *Python* terkenal akan penggunaan *whitespace* dan kesederhanaannya, yang membuat pengguna dapat fokus menyelesaikan masalah dan menciptakan algoritma, bukan memusingkan soal bahasa.

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.

Contoh masukan dari bahasa Python :

```
nama = input("Input Your Name: ")
```

Contoh keluaran dari bahasa Python :

```
print ("Hello ", nama, "!")
```

B. *Syntax Python yang Perlu Diperhatikan*

Python memiliki syntax yang perlu diperhatikan dalam pembuatan CFG. Contohnya adalah:

- Penamaan Variabel

Dalam penamaan variabel, karakter diharuskan diawali dengan suatu huruf, baik kapital maupun kecil, kemudian dibebaskan karakter berikutnya, dengan ini kami mengatur regex untuk variabel (ID) sebagai `[A-Za-z_][A-Za-z0-9_]*`

- Komentar Multibaris

Dalam python, kita dapat menulis komentar multibaris dengan menulis simbol tanda kutip petik satu maupun dua sebanyak 3 kali di awal dan akhir komentar. Karena sistem tokenizer yang mengabaikan newline maupun spasi, kita hanya perlu mengatur regex untuk komentar multibaris menjadi `'''[^']*'''` dan `"""[^\n]*"""`

- Deklarasi If-Elif-Else

Penggunaan conditional dalam python memiliki format sendiri dengan apapun yang ingin dijalankan apabila memenuhi kriteria tertentu diberi sebuah tab. Contoh formatnya seperti ini:

```
if a == 2:
    print("Angkanya 2")
elif a==3:
    print("Angkanya 3")
else:
    print("Angkanya bukan 2 maupun 3")
```

Kami memiliki bagian CFG yang akan menangani kasus conditional branching di python dalam bagian produksi non-terminal

- Deklarasi List

Dalam python, list dibuat dengan mendeklarasikan elemen-elemen di dalam kurung siku, contohnya seperti `[1, 2, 3]`. List dapat dibentuk dengan berbagai macam tipe data seperti Integer `[1,2]`, Float `[1.2, 2.3]`, String `['hey', 'there']`, dst. Kami memiliki bagian CFG yang akan menangani kasus deklarasi list di python dalam bagian produksi non-terminal

- Kalimat Import

Dalam python, kita dapat meng-import modul ke dalam workspace yang kita kerjakan dengan menggunakan keyword import. Cara penulisan import adalah `import (nama modul)`, dengan ini kita bisa memakai fungsi-fungsi yang terdapat di modul tersebut dengan menulis `(nama modul).(fungsi dalam modul tersebut)(kriteria input fungsi)`. Apabila ingin langsung menggunakan fungsi dalam modul, kita bisa memakai `import (fungsi modul) from (nama modul)`. Kami memiliki bagian CFG yang akan menangani

kasus tersebut dalam bagian produksi non-terminal

C. Context Free Grammar

Context Free Language adalah sebuah bahasa yang memiliki cakupan lebih luas daripada *regular language*, yang menggunakan notasi rekursif bernama *Context Free Grammar* (CFG). CFG memiliki empat komponen penting:

1. Terminal (T), berisi simbol-simbol yang akan menjadi *string* dalam sebuah bahasa
2. Variabel atau *non-terminal* (V) yang mendefinisikan satu set bahasa
3. *Start symbol* (S) adalah salah satu dari variabel yang akan dipanggil pertama kali
4. Aturan produksi (P) merupakan definisi rekursif suatu bahasa. Suatu aturan produksi memiliki bentuk:

$$A \rightarrow X$$

dimana A (ruas kiri) adalah sebuah variabel dan X dapat berupa terminal atau variabel lagi yang akan membentuk sebuah *string*.

Komponen CFG biasanya dinotasikan dengan *four-tuple* sebagai

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

Misalnya, *grammar* $G = (\{q\}, \{0, 1\}, A, q)$ memiliki sebuah variabel q , dua terminal berupa 0 dan 1, aturan produksi A , dan *start symbol* q . *Grammar* kemudian dapat digunakan untuk menentukan apakah sebuah *string* termasuk dalam suatu *language* atau tidak. Salah satu cara untuk mengetahui hal tersebut dengan menggunakan *parse tree* yang menggambarkan penurunan suatu *string*.

Parsing

CFG menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator pada umumnya didefinisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree/parse tree*) berfungsi untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel yang akan diturunkan menjadi terminal sampai tidak ada yang belum tergantikan.

Proses penurunan/parsing bisa dilakukan dengan cara sebagai berikut:

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

Misalkan terdapat *grammar* sebagai berikut:

$$\begin{aligned} S &\rightarrow aSA \mid a \\ A &\rightarrow SAb \mid ba \end{aligned}$$

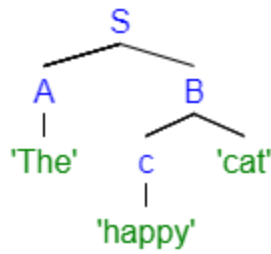
Untuk memperoleh string 'aabba' dari *grammar* diatas dapat dilakukan dengan cara:

- *Leftmost derivation* : $S \Rightarrow aSA \Rightarrow aaSA \Rightarrow aaaA \Rightarrow aaaSAb \Rightarrow aabba$
- *Rightmost derivation* : $S \Rightarrow aSA \Rightarrow aSSAb \Rightarrow aSSbba \Rightarrow aSabba \Rightarrow aabba$

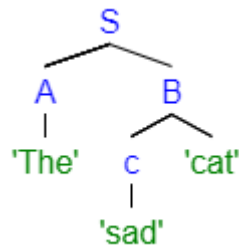
Lalu, berikut contoh CFG sederhana.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow \text{'The'} \\ B &\rightarrow C \text{'cat'} \\ C &\rightarrow \text{'happy'} \mid \text{'sad'} \end{aligned}$$

Dapat dilihat terdapat dua kalimat valid yang akan divisualisasikan dengan *parse tree*, yaitu *the happy cat* dan *the sad cat*.



Gambar 1.1 Analisis kalimat pada Contoh CFG dengan *Parse Tree*



Gambar 1.2 Analisis kalimat pada Contoh CFG dengan *Parse Tree*

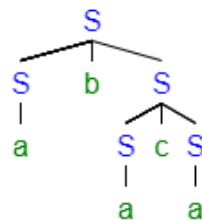
Ambiguitas

Ambiguitas terjadi jika ditemukan lebih dari satu pohon penurunan yang berbeda dalam memperoleh suatu *string* yang sama.

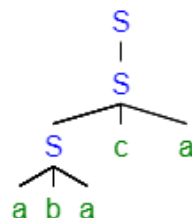
Misalkan terdapat *grammar* sebagai berikut.

$$S \rightarrow SbS \mid ScS \mid a$$

Untuk menurunkan untai 'abaca' dengan *parse tree* diperoleh dua cara, yaitu



Gambar 1.3 Analisis kalimat 'abaca' dengan *Parse Tree*



Gambar 1.4 Analisis kalimat 'abaca' dengan *Parse Tree*

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

D. *Chomsky Normal Form*

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk *Context Free Grammar* (CFG). *Chomsky Normal Form* dapat dibuat dari sebuah *context free grammar* yang telah mengalami penyederhanaan, yaitu penghilangan produksi *useless*, unit, dan ϵ . Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi *chomsky normal form* dengan syarat tata bahasa bebas konteks tersebut:

1. Tidak memiliki produksi *useless*
2. Tidak memiliki produksi unit
3. Tidak memiliki produksi ϵ

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

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

E. *Cocke—Younger—Kasami*

Di dalam dunia ilmu komputer, algoritma *Cocke—Younger—Kasami* (CYK) adalah algoritma parsing untuk *context free grammar*.

Versi standar dari CYK hanya dapat dijalankan pada *context free grammars* yang ditulis dalam *Chomsky Normal Form* (CNF). Bagaimanapun juga, *context free grammar* apapun dapat diubah ke dalam format CNF yang mengekspresikan bahasa yang sama (Sipser 1997).

Algoritma CYK menjadi cukup penting dikarenakan efisiensinya yang tinggi dalam berbagai situasi. Hal ini menjadikan algoritma CYK sebagai salah satu algoritma parsing yang paling efisien dalam hal kompleksitas asimptotik kasus terburuk, meskipun ada algoritma lain dengan waktu berjalan rata-rata yang lebih baik dalam banyak skenario praktis.

Contoh grammar:

$S \rightarrow NP VP$
 $VP \rightarrow VP PP$
 $VP \rightarrow V NP$
 $VP \rightarrow \text{eats}$
 $PP \rightarrow P NP$
 $NP \rightarrow \text{Det } N$
 $NP \rightarrow \text{she}$
 $V \rightarrow \text{eats}$
 $P \rightarrow \text{with}$
 $N \rightarrow \text{fish}$
 $N \rightarrow \text{fork}$
 $\text{Det} \rightarrow \text{a}$

Gambar 1.3 Contoh Grammar

Kalimat *she eats a fish with a fork* dianalisis menggunakan CYK.

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

Gambar 1.2 Analisis kalimat dengan CYK

ANALISIS PERSOALAN

Secara umum, untuk dapat menyelesaikan tugas besar (*compiler python*), kami terlebih dahulu membuat *grammar* dalam CFG yang memuat tata bahasa dan kata kunci bawaan *python* yang didefinisikan di dalam spesifikasi tugas besar. Hal ini bertujuan agar lebih mudah mengimplementasikan bahasa Python ke dalam aturan grammar.

Kata kunci bawaan Python yang diimplementasi dalam compiler ini dikelompokkan ke dalam beberapa kategori, yaitu:

1. Deklarasi, meliputi `def`, `class`, `from`, `import`, `as`;
2. Analisis kasus, meliputi `if`, `elif`, `else`, `is`;
3. Kontrol pengulangan, meliputi `for`, `while`, `in`, `break`, `continue`, `pass`;
4. Operasi logika dan nilainya, meliputi `True`, `False`, `and`, `or`, `not`;
5. Lain-lain, meliputi `with`, `raise`, `none`, `return`.

A. CFG

Untuk membuat sebuah CFG, kami menentukan komponen-komponen penyusunnya terlebih dahulu, yaitu Terminal, Non Terminal, Start Symbol, dan Production Rule. Pada laporan ini kami tidak mencantumkan production rule dari CFG yang kami buat karena kurang efektif apabila ditulis semua. Production rule yang kami buat dapat dilihat pada folder yang akan dikumpulkan pada pranala tugas besar ini.

1. Terminal

Terminal pada CFG kami terdiri dari 77 buah yang mendefinisikan kata kunci. Berikut tabel *Terminal*.

Tabel *Terminal*

COMMENT	COMMENT2	OLCOMMENT	TYPE_FLOAT	TYPE_INT	TYPE_STR	TYPE_STR2	BOOL_TRUE	BOOL_FALSE
TYPE_NONE	TYPEH_DICT	TYPEH_LIST	TYPEH_STRING	TYPEH_FLOAT	TYPEH_BOOL	TYPEH_BYTES	TYPEH_TO	ASSOP_PLUS
ASSOP_MIN	ASSOP_MULT	ASSOP_DIV	ASSOP_MOD	ASSOP_FLOOR	ASSOP_EXP	OP_FLOOR	OP_EXP	OP_PLUS
OP_MULT	OP_DIV	OP_MOD	BINOP_NEGATE	BINOP_XOR	BINOP_LEFTSHIFT	BINOP_RIGHTSHIFT	DOT	COMMA
LOGIC_NOT	LOGIC_AND	LOGIC_OR	LOGIC_AND2	LOGIC_NOT2	LOGIC_OR2	COMP_EQ	COMP_NEQ	COMP_GREATER_EQ
COMP_LESS_EQ	COMP_GREATER	COMP_LESS	ASSOP	OPEN_PAREN	CLOSE_PAREN	OPEN_BRACKET	CLOSE_BRACKET	OPEN_CBRACKET
CLOSE_CBRACKET	FROM	IMPORT	AS	IN	IS	BREAK	CONT	CLASS
DEF	PASS	RETURN	IF	ELIF	ELSE	FOR	WHILE	RAISE
WITH	COLON	ID	TYPE_INT	OP_MIN				

2. Non-Terminal

Non-terminal pada CFG kami terdiri dari 79 buah yang merupakan set bahasa yang dapat dibentuk berdasarkan bahasa *Python*. Berikut tabel *Non-Terminal*.

Tabel *Non-Terminal*

S	IMPORT_METHOD	EXPRESSION	NUM	OBJECT	STRING	FLOAT	NONE	BOOLEAN
TYPE_HINTING	TYPE_HINTING_TO	TYPEH	OP	COMP	ASSIGNMENT	ASSOP	SIGN	COMP

BINOP	STROP	BOOLBINOP	EXP_COMP_EXP	EXP_ASSIGN_EXP	OPEN_PAREN	CLOSE_PAREN	OPEN_BRACKET	CLOSE_BRACKET
OPEN_CBRACKET	CLOSE_CBRACKET	DOT	DOT_OBJ	SEPARATOR	FROM	IMPORT	AS	IN
IS	CLASS	DEF	PASS	RETURN	IF	ELIF	ELSE	FOR
WHILE	RAISE	COLON	VAR_ASSIGNMENT	DEF_METHOD	RETURN_METHOD	IF_METHOD	ELIF_METHOD	ELSE_METHOD
FOR_METHOD	WHILE_METHOD	VAR_METHOD	WITH_METHOD	CLASS_METHOD	LOOP_BREAK	LOOP_CONTINUE	RAISE_METHOD	FUNC
TYPE_HINTING_COLON	OBJ_DOT_OBJ	OBJ_IN_FUNC	OBJ_IN_OBJ	FUNC_AS_OBJ	FRIN_OBJ	IMPORT_OBJ	AS_OBJ	IN_PAREN
IN_BRACKET	IN_CBRACKET	EXPRESSION_IN_BRACKET	EXPRESSION_IN_PAREN	SEPARATOR_EXP	INTEGER	TYPEH_TO		

3. Start Symbol

Start symbol dari *grammar* kami yaitu S, dimana S nantinya akan kemudian memproses kalimat-kalimat yang mungkin dibuat dalam *python*. Beberapa contoh dari kalimat tersebut adalah import, assignment, def, return, conditionals, loop, class, dan expression.

4. Production

Berikut tabel *Production*.

Tabel *Production*

S -> IMPORT_METHOD	TYPEH -> TYPEH_STR	COLON -> COLON
S -> EXPRESSION	TYPEH -> TYPEH_FLOAT	VAR_ASSIGNMENT -> OBJECT ASSOP EXPRESSION
S -> OBJECT	TYPEH -> TYPEH_BOOL	VAR_ASSIGNMENT -> OBJECT ASSIGNMENT EXPRESSION
S -> VAR_ASSIGNMENT	TYPEH -> TYPEH_BYTES	DEF_METHOD -> DEF FUNC COLON
S -> DEF_METHOD	TYPEH_TO -> TYPEH_TO	DEF_METHOD -> DEF FUNC
S -> RETURN_METHOD	OP -> OP_PLUS	TYPE_HINTING_COLON
S -> IF_METHOD	OP -> OP_MIN	RETURN_METHOD -> RETURN EXPRESSION
S -> ELIF_METHOD	OP -> OP_MULT	IF_METHOD -> IF EXP_COMP_EXP COLON
S -> ELSE_METHOD	OP -> OP_DIV	IF_METHOD -> IF IN_PAREN COLON
S -> FOR_METHOD	OP -> OP_MOD	IF_METHOD -> IF OBJECT COLON
S -> WHILE_METHOD	OP -> OP_FLOOR	IF_METHOD -> IF BOOLEAN
S -> VAR_METHOD	OP -> OP_EXP	IF_METHOD -> IF OBJ_IN_OBJ COLON
S -> WITH_METHOD	COMP -> COMP_EQ	ELIF_METHOD -> ELIF EXP_COMP_EXP COLON
S -> CLASS_METHOD	COMP -> COMP_NEQ	ELIF_METHOD -> ELIF IN_PAREN COLON
S -> LOOP_BREAK	COMP -> COMP_GREATER_EQ	ELIF_METHOD -> ELIF OBJECT COLON
S -> LOOP_CONTINUE	COMP -> COMP_LESS_EQ	ELIF_METHOD -> ELIF OBJ_IN_OBJ COLON
S -> PASS	COMP -> COMP_GREATER	ELSE_METHOD -> ELSE COLON
S -> RAISE_METHOD	COMP -> COMP_LESS	FOR_METHOD -> FOR OBJ_IN_OBJ COLON
S -> FUNC	COMP -> IS	WHILE_METHOD -> WHILE IN_PAREN COLON
S -> S S	TYPEH -> TYPEH_INT	WHILE_METHOD -> WHILE EXPRESSION COLON
S -> FROM OBJECT IMPORT OBJECT IMPORT OBJECT	ASSIGNMENT -> ASSOP	VAR_METHOD -> OBJECT DOT FUNC
IMPORT_METHOD -> FROM_OBJ	ASSOP -> ASSOP_PLUS	VAR_METHOD -> OBJECT ASSOP OBJECT
IMPORT_OBJ AS_OBJ FROM_OBJ	ASSOP -> ASSOP_MINUS	VAR_METHOD -> OBJECT ASSIGNMENT OBJECT
IMPORT_OBJ IMPORT_OBJ AS_OBJ IMPORT_OBJ	ASSOP -> ASSOP_MULT	WITH_METHOD -> WITH FUNC_AS_OBJ COLON
EXPRESSION -> STRING	ASSOP -> ASSOP_DIV	CLASS_METHOD -> CLASS FUNC COLON
EXPRESSION -> NUM	ASSOP -> ASSOP_MOD	CLASS_METHOD -> CLASS OBJ COLON
EXPRESSION -> BOOLEAN	ASSOP -> ASSOP_FLOOR	CLASS_METHOD -> CLASS FUNC
EXPRESSION -> OBJECT	ASSOP -> ASSOP_EXP	TYPE_HINTING_COLON
EXPRESSION -> EXPRESSION COMP	SIGN -> OP_PLUS	CLASS_METHOD -> CLASS OBJ
EXPRESSION	SIGN -> OP_MINUS	TYPE_HINTING_COLON
	COMP -> LOGIC_AND	LOOP_BREAK -> BREAK

EXPRESSION -> EXPRESSION OP EXPRESSION EXPRESSION -> FUNC EXPRESSION -> OBJ_DOT_OBJ EXPRESSION -> OBJ_DOT_FUNC EXPRESSION -> IN_BRACKET EXPRESSION -> IN_PAREN EXPRESSION -> IN_CBRACKET EXPRESSION -> NUM OP NUM EXPRESSION -> STRING OP_MULTIPLY INTEGER EXPRESSION -> STRING OP_PLUS STRING EXPRESSION -> OBJECT OP OBJECT EXPRESSION -> BINOP NUM EXPRESSION -> BOOLBINOP BOOLEAN EXPRESSION -> BINOP OBJECT NUM -> INTEGER NUM -> SIGN INTEGER NUM -> FLOAT NUM -> SIGN FLOAT OBJECT -> OBJECT DOT OBJECT OBJECT -> OBJECT SEPARATOR OBJECT OBJECT -> OBJECT DOT FUNC OBJECT -> OBJECT IS OBJECT OBJECT -> OBJECT AS OBJECT OBJECT -> OBJECT IN OBJECT OBJECT -> OBJECT IN_BRACKET DOT_OBJ OBJECT -> OBJECT IN_BRACKET OBJECT -> OBJECT IN_PAREN OBJECT -> OBJECT IN_PAREN DOT_OBJ OBJECT -> ID STRING -> STRING DOT STRING STRING -> STRING DOT OBJECT STRING -> TYPE_STR STRING -> TYPE_STR2 INTEGER -> TYPE_INT FLOAT -> TYPE_FLOAT NONE -> TYPE_NONE BOOLEAN -> BOOL_TRUE BOOLEAN -> BOOL_FALSE TYPE_HINTING -> COLON TYPEH TYPE_HINTING_TO -> TYPEH_TO TYPEH TYPEH -> TYPEH_DICT TYPEH -> TYPEH_LIST	COMP -> LOGIC_OR COMP -> LOGIC_AND2 COMP -> LOGIC_OR2 BINOP -> LOGIC_NOT BINOP -> LOGIC_NOT2 BINOP -> BINOP_XOR BINOP -> BINOP_LEFTSHIFT BINOP -> BINOP_RIGHTSHIFT STROP -> OP_PLUS STROP -> OP_MULTIPLY BOOLBINOP -> BINOP_NEGATE BOOLBINOP -> LOGIC_NOT BOOLBINOP -> LOGIC_NOT2 EXP_COMP_EXP -> EXPRESSION COMP EXPRESSION EXP_ASSIGN_EXP -> EXPRESSION ASSIGNMENT EXPRESSION EXP_ASSIGN_EXP -> EXPRESSION ASSOP EXPRESSION OPEN_PAREN -> OPEN_PAREN CLOSE_PAREN -> CLOSE_PAREN OPEN_BRACKET -> OPEN_BRACK CLOSE_BRACKET -> CLOSE_BRACK OPEN_CBRACKET -> OPEN_CBRACK CLOSE_CBRACKET -> CLOSE_CBRACK DOT -> DOT DOT_OBJ -> DOT OBJECT SEPARATOR -> COMMA FROM -> FROM IMPORT -> IMPORT AS -> AS IN -> IN IS -> IS CLASS -> CLASS DEF -> DEF PASS -> PASS RETURN -> RETURN IF -> IF ELIF -> ELIF ELSE -> ELSE FOR -> FOR WHILE -> WHILE RAISE -> RAISE FOR_METHOD -> FOR OBJ_IN_FUNC COLON	LOOP_CONTINUE -> CONT RAISE_METHOD -> RAISE OBJECT FUNC -> OBJECT IN_PAREN TYPE_HINTING_COLON -> TYPE_HINTING_TO COLON OBJ_DOT_OBJ -> OBJ_DOT OBJ_DOT OBJ_IN_FUNC -> OBJECT IN FUNC OBJ_IN_OBJ -> OBJECT IN OBJECT FUNC_AS_OBJ -> FUNC AS OBJECT FROM_OBJ -> FROM OBJECT IMPORT_OBJ -> IMPORT OBJECT AS_OBJ -> AS OBJECT IN_PAREN -> OPEN_PAREN EXPRESSION_IN_PAREN CLOSE_PAREN IN_PAREN -> OPEN_PAREN CLOSE_PAREN IN_BRACKET -> OPEN_BRACKET EXPRESSION_IN_BRACKET CLOSE_BRACKET IN_BRACKET -> OPEN_BRACKET CLOSE_BRACKET IN_CBRACKET -> OPEN_CBRACKET CLOSE_CBRACKET EXPRESSION_IN_BRACKET -> EXPRESSION EXPRESSION_IN_BRACKET -> TYPE_HINTING EXPRESSION_IN_PAREN -> EXPRESSION EXPRESSION_IN_PAREN -> EXP_COMP_EXP EXPRESSION_IN_PAREN -> EXP_ASSIGN_EXP EXPRESSION_IN_PAREN -> EXPRESSION TYPE_HINTING EXPRESSION_IN_PAREN -> EXPRESSION TYPE_HINTING SEPARATOR_EXP EXPRESSION_IN_PAREN -> EXPRESSION SEPARATOR_EXP SEPARATOR_EXP -> SEPARATOR EXPRESSION_IN_PARENEXPRESSSION_IN_BRACKET -> NUM SEPARATOR EXPRESSION_IN_BRACKET NUM
--	--	--

CFG yang kami buat nantinya akan dikonversi menjadi CNF agar dapat dijadikan parameter masukan CYK Parser.

B. CNF

Berikut adalah hasil CFG yang sudah menjadi bentuk CNF yang telah didapatkan setelah dikonversi.

1. Terminal

Terminal pada CNF kami terdiri dari 77 buah yang mendefinisikan kata kunci. Berikut tabel *Terminal*.

Tabel *Terminal*

COMMENT	COMMENT2	OLCOMM ENT	TYPE_FLOA T	TYPE_INT	TYPE_STR	TYPE_STR2	BOOL_TRU E	BOOL_FALSE

TYPE_NONE	TYPEH_DICT	TYPEH_LIST	TYPEH_STRING	TYPEH_FLOAT	TYPEH_BOOLEAN	TYPEH_BYTES	TYPEH_TO	ASSOP_PLUS
ASSOP_MIN	ASSOP_MULT	ASSOP_DIV	ASSOP_MOD	ASSOP_FLOOR	ASSOP_EXP	OP_FLOOR	OP_EXP	OP_PLUS
OP_MULT	OP_DIV	OP_MOD	BINOP_NEGATE	BINOP_XOR	BINOP_LEFTSHIFT	BINOP_RIGHTSHIFT	DOT	COMMA
LOGIC_NOT	LOGIC_AND	LOGIC_OR	LOGIC_AND2	LOGIC_NOT2	LOGIC_OR2	COMP_EQ	COMP_NEQ	COMP_GREATER_EQ
COMP_LESS_EQ	COMP_GREATER	COMP_LESS	ASSOP	OPEN_PAREN	CLOSE_PAREN	OPEN_BRACKET	CLOSE_BRACKET	OPEN_CBRACKET
CLOSE_CBRACKET	FROM	IMPORT	AS	IN	IS	BREAK	CONT	CLASS
DEF	PASS	RETURN	IF	ELIF	ELSE	FOR	WHILE	RAISE
WITH	COLON	ID	TYPE_INT	OP_MIN				

2. Non-Terminal

Non-terminal pada CNF kami terdiri dari 79 buah yang merupakan set bahasa yang dapat dibentuk berdasarkan bahasa *Python*. Berikut tabel *Non-Terminal*.

Tabel *Non-Terminal*

S	IMPORT_METHOD	EXPRESSION	NUM	OBJECT	STRING	FLOAT	NONE	BOOLEAN
TYPE_HINTING	TYPE_HINTING_TO	TYPEH	OP	COMP	ASSIGNMENT	ASSOP	SIGN	COMP
BINOP	STROP	BOOLBINOP	EXP_COMP_EXP	EXP_ASSIGN_EXP	OPEN_PAREN	CLOSE_PAREN	OPEN_BRACKET	CLOSE_BRACKET
OPEN_CBRACKET	CLOSE_CBRACKET	DOT	DOT_OBJ	SEPARATOR	FROM	IMPORT	AS	IN
IS	CLASS	DEF	PASS	RETURN	IF	ELIF	ELSE	FOR
WHILE	RAISE	COLON	VAR_ASSIGNMENT	DEF_METHOD	RETURN_METHOD	IF_METHOD	ELIF_METHOD	ELSE_METHOD
FOR_METHOD	WHILE_METHOD	VAR_METHOD	WITH_METHOD	CLASS_METHOD	LOOP_BREAK	LOOP_CONTINUE	RAISE_METHOD	FUNC
TYPE_HINTING_COLON	OBJ_DOT_OBJ	OBJ_IN_FUNC	OBJ_IN_OBJ	FUNC_AS_OBJ	FRIN_OBJ	IMPORT_OBJ	AS_OBJ	IN_PAREN
IN_BRACKET	IN_CBRACKET	EXPRESSION_IN_BRACKET	EXPRESSION_IN_PAREN	SEPARATOR_EXP	INTEGER	TYPEH_TO		

3. Start Symbol

Start symbol dari *grammar* kami yaitu S, dimana S nantinya akan kemudian memproses kalimat-kalimat yang mungkin dibuat dalam *python*. Beberapa contoh dari kalimat tersebut adalah import, assignment, def, return, conditionals, loop, class, dan expression.

4. Production

Berikut tabel *Production*.

Tabel *Production*

IMPORT_METHOD -> IMPORT_METHOD8 IMPORT_OBJ EXPRESSION -> EXPRESSION9 EXPRESSION EXPRESSION9 -> EXPRESSION COMP EXPRESSION -> EXPRESSION10 EXPRESSION	VAR_METHOD -> VAR_METHOD44 FUNC VAR_METHOD44 -> OBJECT DOT VAR_METHOD -> VAR_METHOD45 OBJECT VAR_METHOD45 -> OBJECT ASSOP VAR_METHOD -> VAR_METHOD46 OBJECT	EXPRESSION_IN_BRACKET -> STRING23 STRING EXPRESSION_IN_BRACKET -> STRING24 OBJECT EXPRESSION_IN_BRACKET -> TYPE_STR
--	---	---

<p> EXPRESSION10 -> EXPRESSION OP EXPRESSION -> EXPRESSION11 NUM EXPRESSION11 -> NUM OP EXPRESSION -> EXPRESSION12 INTEGER EXPRESSION12 -> STRING OP_MULTIPLY EXPRESSION -> EXPRESSION13 STRING EXPRESSION13 -> STRING OP_PLUS EXPRESSION -> EXPRESSION14 OBJECT EXPRESSION14 -> OBJECT OP EXPRESSION -> BINOP NUM EXPRESSION -> BOOLBINOP BOOLEAN EXPRESSION -> BINOP OBJECT NUM -> SIGN INTEGER NUM -> SIGN FLOAT OBJECT -> OBJECT15 OBJECT OBJECT15 -> OBJECT DOT OBJECT -> OBJECT16 OBJECT OBJECT16 -> OBJECT SEPARATOR OBJECT -> OBJECT17 FUNC OBJECT17 -> OBJECT DOT OBJECT -> OBJECT18 OBJECT OBJECT18 -> OBJECT IS OBJECT -> OBJECT19 OBJECT OBJECT19 -> OBJECT AS OBJECT -> OBJECT20 OBJECT OBJECT20 -> OBJECT IN OBJECT -> OBJECT21 DOT_OBJ OBJECT21 -> OBJECT IN_BRACKET OBJECT -> OBJECT IN_BRACKET OBJECT -> OBJECT IN_PAREN OBJECT -> OBJECT22 DOT_OBJ OBJECT22 -> OBJECT IN_PAREN OBJECT -> ID STRING -> STRING23 STRING STRING23 -> STRING DOT STRING -> STRING24 OBJECT STRING24 -> STRING DOT STRING -> TYPE_STR STRING -> TYPE_STR2 INTEGER -> TYPE_INT FLOAT -> TYPE_FLOAT NONE -> TYPE_NONE BOOLEAN -> BOOL_TRUE BOOLEAN -> BOOL_FALSE TYPE_HINTING -> COLON TYPEH TYPE_HINTING_TO -> TYPEH_TO TYPEH TYPEH -> TYPEH_DICT TYPEH -> TYPEH_LIST TYPEH -> TYPEH_INT TYPEH -> TYPEH_STR TYPEH -> TYPEH_FLOAT TYPEH -> TYPEH_BOOL TYPEH -> TYPEH_BYTES TYPEH_TO -> TYPEH_TO OP -> OP_PLUS OP -> OP_MIN </p>	<p> VAR_METHOD46 -> OBJECT ASSIGNMENT WITH_METHOD -> WITH_METHOD47 COLON WITH_METHOD47 -> WITH_FUNC_AS_OBJ CLASS_METHOD -> CLASS_METHOD48 COLON CLASS_METHOD48 -> CLASS FUNC CLASS_METHOD -> CLASS_METHOD49 COLON CLASS_METHOD49 -> CLASS OBJ CLASS_METHOD -> CLASS_METHOD50 TYPE_HINTING_COLON CLASS_METHOD50 -> CLASS FUNC CLASS_METHOD -> CLASS_METHOD51 TYPE_HINTING_COLON CLASS_METHOD51 -> CLASS OBJ LOOP_BREAK -> BREAK LOOP_CONTINUE -> CONT RAISE_METHOD -> RAISE OBJECT FUNC -> OBJECT IN_PAREN TYPE_HINTING_COLON -> TYPE_HINTING_TO COLON OBJ_DOT_OBJ -> OBJ_DOT OBJ_DOT OBJ_IN_FUNC -> OBJ_IN_FUNC52 FUNC OBJ_IN_FUNC52 -> OBJECT IN OBJ_IN_OBJ -> OBJ_IN_OBJ53 OBJECT OBJ_IN_OBJ53 -> OBJECT IN FUNC_AS_OBJ -> FUNC_AS_OBJ54 OBJECT FUNC_AS_OBJ54 -> FUNC AS FROM_OBJ -> FROM OBJECT IMPORT_OBJ -> IMPORT OBJECT AS_OBJ -> AS OBJECT IN_PAREN -> IN_PAREN55 CLOSE_PAREN IN_PAREN55 -> OPEN_PAREN EXPRESSION_IN_PAREN IN_PAREN -> OPEN_PAREN CLOSE_PAREN IN_BRACKET -> IN_BRACKET56 CLOSE_BRACKET IN_BRACKET56 -> OPEN_BRACKET EXPRESSION_IN_BRACKET IN_BRACKET -> OPEN_BRACKET CLOSE_BRACKET IN_CBRACKET -> OPEN_CBRACKET CLOSE_CBRACKET EXPRESSION_IN_PAREN -> EXPRESSION TYPE_HINTING EXPRESSION_IN_PAREN -> EXPRESSION_IN_PAREN57 SEPARATOR_EXP EXPRESSION_IN_PAREN57 -> EXPRESSION TYPE_HINTING EXPRESSION_IN_PAREN -> EXPRESSION SEPARATOR_EXP SEPARATOR_EXP -> SEPARATOR EXPRESSION_IN_PAREN EXPRESSION_IN_PAREN -> EXP_ASSIGN_EXP26 EXPRESSION EXPRESSION_IN_PAREN -> EXP_ASSIGN_EXP27 EXPRESSION EXPRESSION_IN_PAREN -> EXP_COMP_EXP25 EXPRESSION </p>	<p> EXPRESSION_IN_BRACKET -> TYPE_STR2 COMP -> IS NUM -> TYPE_FLOAT NUM -> TYPE_INT EXPRESSION -> OPEN_CBRACKET CLOSE_CBRACKET EXPRESSION -> IN_PAREN55 CLOSE_PAREN EXPRESSION -> OPEN_PAREN CLOSE_PAREN EXPRESSION -> IN_BRACKET56 CLOSE_BRACKET EXPRESSION -> OPEN_BRACKET CLOSE_BRACKET EXPRESSION -> OBJ_DOT OBJ_DOT EXPRESSION -> OBJECT IN_PAREN EXPRESSION -> OBJECT15 OBJECT EXPRESSION -> OBJECT16 OBJECT EXPRESSION -> OBJECT17 FUNC EXPRESSION -> OBJECT18 OBJECT EXPRESSION -> OBJECT19 OBJECT EXPRESSION -> OBJECT20 OBJECT EXPRESSION -> OBJECT21 DOT_OBJ EXPRESSION -> OBJECT IN_BRACKET EXPRESSION -> OBJECT IN_PAREN EXPRESSION -> OBJECT22 DOT_OBJ EXPRESSION -> ID EXPRESSION -> BOOL_TRUE EXPRESSION -> BOOL_FALSE EXPRESSION -> SIGN INTEGER EXPRESSION -> SIGN FLOAT EXPRESSION -> TYPE_FLOAT EXPRESSION -> TYPE_INT EXPRESSION -> TYPE_FLOAT EXPRESSION -> TYPE_INT EXPRESSION -> STRING23 STRING EXPRESSION -> STRING24 OBJECT EXPRESSION -> TYPE_STR EXPRESSION -> TYPE_STR2 S -> S S S -> OBJECT IN_PAREN S -> RAISE OBJECT S -> PASS S -> CONT S -> BREAK S -> CLASS_METHOD48 COLON S -> CLASS_METHOD49 COLON S -> CLASS_METHOD50 TYPE_HINTING_COLON S -> CLASS_METHOD51 TYPE_HINTING_COLON S -> WITH_METHOD47 COLON S -> VAR_METHOD44 FUNC S -> VAR_METHOD45 OBJECT S -> VAR_METHOD46 OBJECT S -> WHILE_METHOD42 COLON S -> WHILE_METHOD43 COLON S -> FOR_METHOD40 COLON S -> FOR_METHOD41 COLON </p>
---	---	---

OP -> OP_MULT	EXPRESSION_IN_PAREN -> EXPRESSION9	S -> ELSE COLON
OP -> OP_DIV	EXPRESSION	S -> ELIF_METHOD36 COLON
OP -> OP_MOD	EXPRESSION_IN_PAREN -> EXPRESSION10	S -> ELIF_METHOD37 COLON
OP -> OP_FLOOR	EXPRESSION	S -> ELIF_METHOD38 COLON
OP -> OP_EXP	EXPRESSION_IN_PAREN -> EXPRESSION11	S -> ELIF_METHOD39 COLON
COMP -> COMP_EQ	NUM	S -> IF_METHOD32 COLON
COMP -> COMP_NEQ	EXPRESSION_IN_PAREN -> EXPRESSION12	S -> IF_METHOD33 COLON
COMP -> COMP_GREATER_EQ	INTEGER	S -> IF_METHOD34 COLON
COMP -> COMP_LESS_EQ	EXPRESSION_IN_PAREN -> EXPRESSION13	S -> IF_METHOD35 COLON
COMP -> COMP_GREATER	STRING	S -> RETURN EXPRESSION
COMP -> COMP_LESS	EXPRESSION_IN_PAREN -> EXPRESSION14	S -> DEF_METHOD30 COLON
ASSIGNMENT -> ASSOP	OBJECT	S -> DEF_METHOD31 TYPE_HINTING_COLON
ASSOP -> ASSOP_PLUS	EXPRESSION_IN_PAREN -> BINOP NUM	S -> VAR_ASSIGNMENT28 EXPRESSION
ASSOP -> ASSOP_MINUS	EXPRESSION_IN_PAREN -> BOOLBINOP	S -> VAR_ASSIGNMENT29 EXPRESSION
ASSOP -> ASSOP_MULT	BOOLEAN	S -> OBJECT15 OBJECT
ASSOP -> ASSOP_DIV	EXPRESSION_IN_PAREN -> BINOP OBJECT	S -> OBJECT16 OBJECT
ASSOP -> ASSOP_MOD	EXPRESSION_IN_PAREN -> OPEN_CBRACKET	S -> OBJECT17 FUNC
ASSOP -> ASSOP_FLOOR	CLOSE_CBRACKET	S -> OBJECT18 OBJECT
ASSOP -> ASSOP_EXP	EXPRESSION_IN_PAREN -> IN_PAREN55	S -> OBJECT19 OBJECT
SIGN -> OP_PLUS	CLOSE_PAREN	S -> OBJECT20 OBJECT
SIGN -> OP_MINUS	EXPRESSION_IN_PAREN -> OPEN_PAREN	S -> OBJECT21 DOT_OBJ
COMP -> LOGIC_AND	CLOSE_PAREN	S -> OBJECT IN_BRACKET
COMP -> LOGIC_OR	EXPRESSION_IN_PAREN -> IN_BRACKET56	S -> OBJECT IN_PAREN
COMP -> LOGIC_AND2	CLOSE_BRACKET	S -> OBJECT22 DOT_OBJ
COMP -> LOGIC_OR2	EXPRESSION_IN_PAREN -> OPEN_BRACKET	S -> ID
BINOP -> LOGIC_NOT	CLOSE_BRACKET	S -> EXPRESSION9 EXPRESSION
BINOP -> LOGIC_NOT2	EXPRESSION_IN_PAREN -> OBJ_DOT	S -> EXPRESSION10 EXPRESSION
BINOP -> BINOP_XOR	OBJ_DOT	S -> EXPRESSION11 NUM
BINOP -> BINOP_LEFTSHIFT	EXPRESSION_IN_PAREN -> OBJECT	S -> EXPRESSION12 INTEGER
BINOP -> BINOP_RIGHTSHIFT	IN_PAREN	S -> EXPRESSION13 STRING
STROP -> OP_PLUS	EXPRESSION_IN_PAREN -> OBJECT15	S -> EXPRESSION14 OBJECT
STROP -> OP_MULTIPLY	OBJECT	S -> BINOP NUM
BOOLBINOP -> BINOP_NEGATE	EXPRESSION_IN_PAREN -> OBJECT16	S -> BOOLBINOP BOOLEAN
BOOLBINOP -> LOGIC_NOT	OBJECT	S -> BINOP OBJECT
BOOLBINOP -> LOGIC_NOT2	EXPRESSION_IN_PAREN -> OBJECT17 FUNC	S -> OPEN_CBRACKET CLOSE_CBRACKET
EXP_COMP_EXP -> EXP_COMP_EXP25	EXPRESSION_IN_PAREN -> OBJECT18	S -> IN_PAREN55 CLOSE_PAREN
EXPRESSION	OBJECT	S -> OPEN_PAREN CLOSE_PAREN
EXP_COMP_EXP25 -> EXPRESSION COMP	EXPRESSION_IN_PAREN -> OBJECT19	S -> IN_BRACKET56 CLOSE_BRACKET
EXP_ASSIGN_EXP -> EXP_ASSIGN_EXP26	OBJECT	S -> OPEN_BRACKET CLOSE_BRACKET
EXPRESSION	EXPRESSION_IN_PAREN -> OBJECT20	S -> OBJ_DOT OBJ_DOT
EXP_ASSIGN_EXP26 -> EXPRESSION	OBJECT	S -> OBJECT IN_PAREN
ASSIGNMENT	EXPRESSION_IN_PAREN -> OBJECT21	S -> OBJECT15 OBJECT
EXP_ASSIGN_EXP -> EXP_ASSIGN_EXP27	DOT_OBJ	S -> OBJECT16 OBJECT
EXPRESSION	EXPRESSION_IN_PAREN -> OBJECT	S -> OBJECT17 FUNC
EXP_ASSIGN_EXP27 -> EXPRESSION ASSOP	IN_BRACKET	S -> OBJECT18 OBJECT
OPEN_PAREN -> OPEN_PAREN	EXPRESSION_IN_PAREN -> OBJECT	S -> OBJECT19 OBJECT
CLOSE_PAREN -> CLOSE_PAREN	IN_PAREN	S -> OBJECT20 OBJECT
OPEN_BRACKET -> OPEN_BRACK	EXPRESSION_IN_PAREN -> OBJECT22	S -> OBJECT21 DOT_OBJ
CLOSE_BRACKET -> CLOSE_BRACK	DOT_OBJ	S -> OBJECT IN_BRACKET
OPEN_CBRACKET -> OPEN_CBRACK	EXPRESSION_IN_PAREN -> ID	S -> OBJECT IN_PAREN
CLOSE_CBRACKET -> CLOSE_CBRACK	EXPRESSION_IN_PAREN -> BOOL_TRUE	S -> OBJECT22 DOT_OBJ
DOT -> DOT	EXPRESSION_IN_PAREN -> BOOL_FALSE	S -> ID
DOT_OBJ -> DOT OBJECT	EXPRESSION_IN_PAREN -> SIGN INTEGER	S -> BOOL_TRUE
SEPARATOR -> COMMA	EXPRESSION_IN_PAREN -> SIGN FLOAT	EXPRESSION_IN_BRACKET -> BOOL_TRUE
FROM -> FROM	EXPRESSION_IN_PAREN -> TYPE_FLOAT	EXPRESSION_IN_BRACKET -> BOOL_FALSE
IMPORT -> IMPORT	EXPRESSION_IN_PAREN -> TYPE_FLOAT	EXPRESSION_IN_BRACKET -> SIGN INTEGER
AS -> AS	EXPRESSION_IN_PAREN -> TYPE_INT	
	EXPRESSION_IN_PAREN -> STRING23 STRING	
	EXPRESSION_IN_PAREN -> STRING24 OBJECT	
	EXPRESSION_IN_PAREN -> TYPE_STR	
	EXPRESSION_IN_PAREN -> TYPE_STR2	

IN -> IN	EXPRESSION_IN_BRACKET -> COLON TYPEH	EXPRESSION_IN_BRACKET -> ID
IS -> IS	EXPRESSION_IN_BRACKET -> EXPRESSION9	EXPRESSION_IN_BRACKET -> SIGN FLOAT
CLASS -> CLASS	EXPRESSION	EXPRESSION_IN_BRACKET -> TYPE_FLOAT
DEF -> DEF	EXPRESSION_IN_BRACKET -> EXPRESSION10	EXPRESSION_IN_BRACKET -> TYPE_INT
PASS -> PASS	EXPRESSION	S -> BOOL_FALSE
RETURN -> RETURN	EXPRESSION_IN_BRACKET -> EXPRESSION11	S -> SIGN INTEGER
IF -> IF	NUM	S -> SIGN FLOAT
ELIF -> ELIF	EXPRESSION_IN_BRACKET -> EXPRESSION15	S -> TYPE_FLOAT
ELSE -> ELSE	NUM	S -> TYPE_INT
FOR -> FOR	EXPRESSION15 -> EXPRESSION16	S -> TYPE_FLOAT
WHILE -> WHILE	SEPARATOR	S -> TYPE_INT
RAISE -> RAISE	EXPRESSION16 -> EXPRESSION15 NUM	S -> TYPE_FLOAT
COLON -> COLON	EXPRESSION15 -> NUM SEPARATOR	S -> STRING23 STRING
VAR_ASSIGNMENT -> VAR_ASSIGNMENT28	EXPRESSION_IN_BRACKET -> EXPRESSION12	S -> STRING24 OBJECT
EXPRESSION	INTEGER	S -> TYPE_STR
VAR_ASSIGNMENT28 -> OBJECT ASSOP	EXPRESSION_IN_BRACKET -> EXPRESSION13	S -> TYPE_STR2
VAR_ASSIGNMENT -> VAR_ASSIGNMENT29	STRING	S -> TYPE_FLOAT
EXPRESSION	EXPRESSION_IN_BRACKET -> EXPRESSION14	S -> TYPE_INT
VAR_ASSIGNMENT29 -> OBJECT	OBJECT	S -> OPEN_CBRACKET CLOSE_CBRACKET
ASSIGNMENT	EXPRESSION_IN_BRACKET -> BINOP NUM	S -> IN_PAREN55 CLOSE_PAREN
DEF_METHOD -> DEF_METHOD30 COLON	EXPRESSION_IN_BRACKET -> BOOLBINOP	S -> OPEN_PAREN CLOSE_PAREN
DEF_METHOD30 -> DEF FUNC	BOOLEAN	S -> IN_BRACKET56 CLOSE_BRACKET
DEF_METHOD -> DEF_METHOD31	EXPRESSION_IN_BRACKET -> BINOP OBJECT	S -> OPEN_BRACKET CLOSE_BRACKET
TYPE_HINTING COLON	EXPRESSION_IN_BRACKET ->	S -> OBJ_DOT OBJ_DOT
DEF_METHOD31 -> DEF FUNC	OPEN_CBRACKET CLOSE_CBRACKET	S -> OBJECT IN_PAREN
RETURN_METHOD -> RETURN EXPRESSION	EXPRESSION_IN_BRACKET -> IN_PAREN55	S -> OBJECT15 OBJECT
IF_METHOD -> IF_METHOD32 COLON	CLOSE_PAREN	S -> OBJECT16 OBJECT
IF_METHOD32 -> IF EXP_COMP_EXP	EXPRESSION_IN_BRACKET -> OPEN_PAREN	S -> OBJECT17 FUNC
IF_METHOD -> IF_METHOD33 COLON	CLOSE_PAREN	S -> OBJECT18 OBJECT
IF_METHOD33 -> IF IN_PAREN	EXPRESSION_IN_BRACKET ->	S -> OBJECT19 OBJECT
IF_METHOD -> IF_METHOD34 COLON	IN_BRACKET56 CLOSE_BRACKET	S -> OBJECT20 OBJECT
IF_METHOD34 -> IF OBJECT IF BOOLEAN	EXPRESSION_IN_BRACKET ->	S -> OBJECT21 DOT_OBJ
IF_METHOD -> IF_METHOD35 COLON	OPEN_BRACKET CLOSE_BRACKET	S -> OBJECT IN_BRACKET
IF_METHOD35 -> IF OBJ_IN_OBJ	EXPRESSION_IN_BRACKET -> OBJ_DOT	S -> OBJECT IN_PAREN
ELIF_METHOD -> ELIF_METHOD36 COLON	OBJ_DOT	S -> OBJECT22 DOT_OBJ
ELIF_METHOD36 -> ELIF EXP_COMP_EXP	EXPRESSION_IN_BRACKET -> OBJECT	S -> ID
ELIF_METHOD -> ELIF_METHOD37 COLON	IN_PAREN	S -> BOOL_TRUE
ELIF_METHOD37 -> ELIF IN_PAREN	EXPRESSION_IN_BRACKET -> OBJECT15	S -> BOOL_FALSE
ELIF_METHOD -> ELIF_METHOD38 COLON	OBJECT	S -> SIGN INTEGER
ELIF_METHOD38 -> ELIF OBJECT	EXPRESSION_IN_BRACKET -> OBJECT16	S -> SIGN FLOAT
ELIF_METHOD -> ELIF_METHOD39 COLON	OBJECT	S -> TYPE_FLOAT
ELIF_METHOD39 -> ELIF OBJ_IN_OBJ	EXPRESSION_IN_BRACKET -> OBJECT17	S -> TYPE_INT
ELSE_METHOD -> ELSE COLON	FUNC	S -> TYPE_FLOAT
FOR_METHOD -> FOR_METHOD40 COLON	EXPRESSION_IN_BRACKET -> OBJECT18	S -> TYPE_INT
FOR_METHOD40 -> FOR OBJ_IN_FUNC	OBJECT	S -> STRING23 STRING
FOR_METHOD -> FOR_METHOD41 COLON	EXPRESSION_IN_BRACKET -> OBJECT19	S -> STRING24 OBJECT
FOR_METHOD41 -> FOR OBJ_IN_OBJ	OBJECT	S -> TYPE_STR
WHILE_METHOD -> WHILE_METHOD42	EXPRESSION_IN_BRACKET -> OBJECT20	S -> TYPE_STR2
COLON	OBJECT	S -> IMPORT_METHOD8 IMPORT_OBJ
WHILE_METHOD42 -> WHILE IN_PAREN	EXPRESSION_IN_BRACKET -> OBJECT21	S -> IMPORT OBJECT
WHILE_METHOD -> WHILE_METHOD43	DOT_OBJ	S -> FROM_OBJ IMPORT_METHOD8
COLON	EXPRESSION_IN_BRACKET -> OBJECT	IMPORT_METHOD8 -> IMPORT OBJECT
WHILE_METHOD43 -> WHILE EXPRESSION	IN_BRACKET	
	EXPRESSION_IN_BRACKET -> OBJECT	
	IN_PAREN	
	EXPRESSION_IN_BRACKET -> OBJECT22	
	DOT_OBJ	

IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

Program dibagi ke 3 bagian yaitu “main.py”, “CYKParser.py”, dan “Tokenizer.py”. Alur program dimulai dengan memasukkan perintah di terminal “py main.py <file yang ingin dicek>”, perintah ini akan memanggil program “main.py” alias program utama yang akan kemudian mencari apakah nama file ada, bila ada lanjutkan ke tahap berikutnya yaitu mengubah isi file tersebut menjadi token-token yang dapat dibaca dengan CFG. Program utama kemudian akan memanggil “Tokenizer.py” untuk mengubah kode di dalam file yang diberikan menjadi token-token menggunakan regex. Setelah diubah, program utama akan menyimpannya dalam file “Tokenized.txt” untuk dipakai pada fase berikutnya. Setelah membuat token, kita akan menghilangkan token-token yang merupakan komentar dari python agar tidak perlu kita cek menggunakan CYK, serta menghilangkan spasi di ujung string untuk membersihkan input. Untuk mengecek apakah token benar, akan dilakukan menggunakan algoritma CYK. Awalnya kita membuat CFG yang akan ditaruh di “grammar.txt”, kemudian mengkonversinya menjadi CNF dengan bantuan converter online. dengan tahap sebagai berikut:

1. Menghilangkan *null production*;
2. Menghilangkan *unit production*;
3. Menghilangkan *useless production*;
4. Dekomposisi RHS bila terdapat terminal dan not-terminal yang tergabung;
5. Dekomposisi RHS bila terdapat lebih dari dua non-terminal;

Program utama kemudian akan mengambil CNF yang telah dibuat untuk dasar dilakukannya CYK. CYK dilakukan dengan memanggil program “CYKParser.py”. Program pertama memanggil fungsi untuk mengubah file “CNF.txt” menjadi suatu list yang bisa digunakan, kemudian program akan menggunakan list tersebut untuk mengubah token menjadi baris pertama dalam tabel segitiga CYK. Kemudian, menggunakan list grammar yang sama, mulai mengerjakan baris-baris berikutnya sampai mencapai baris paling atas. Kemudian program akan mengecek apakah di sel paling atas terdapat ‘S’ atau tidak, jika iya maka bisa dibilang file yang dimasukkan dapat di-compile sehingga akan mengeluarkan output “Accepted”. Jika tidak ada ‘S’ maka program akan mengeluarkan output “Syntax Error”.

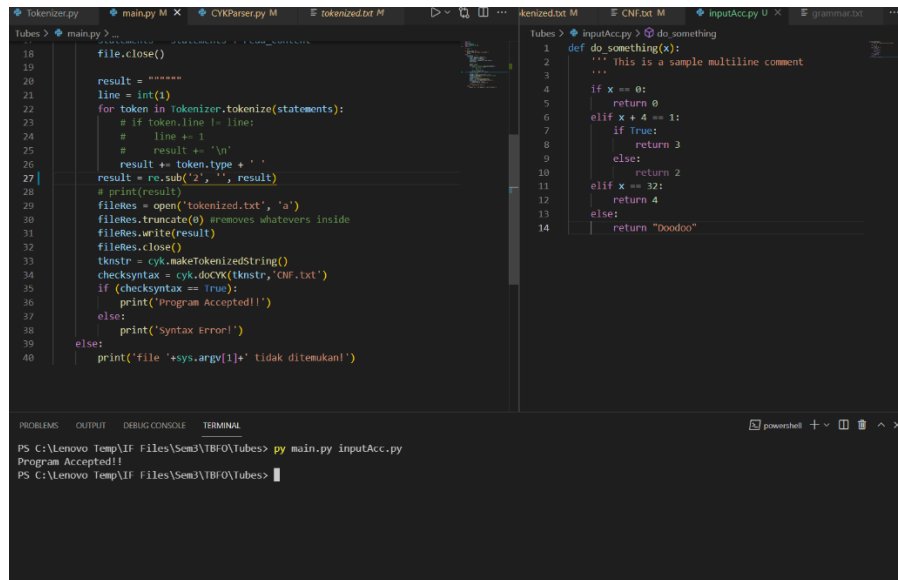
1. Program memiliki beberapa struktur data, antara lain:
 - Grammar berupa List of List
 - cykArray berupa List of List of Sets. Sets dipakai di sini karena sifatnya yang tidak memiliki duplikat, agar apabila terdapat Left Hand Side Terminal yang sama berulang kali, kita hanya perlu mengecek satu saja.
2. Terdapat beberapa fungsi dan prosedur dalam program yang akan dipanggil, antara lain:
 - Fungsi Tokenize yang menghasilkan token dari suatu string
 - Fungsi makeListOfGrammar yang menghasilkan list of grammar dari suatu file yang berisi grammar CNF
 - Fungsi zerothLine yang menghasilkan baris pertama dari tabel CYK dari suatu string token
 - Fungsi makeTokenizedString yang menghasilkan string token dari suatu file yang berisi token
 - Fungsi doCYK yang menghasilkan True apabila hasil CYK merupakan ‘S’ dengan input string token dan file grammar CNF.

3. Antarmuka

Program memiliki antarmuka yang simple berbasis CLI, kita memanggil program dengan menjalankan terminal di folder yang sama dengan program, dan melakukan perintah “py main.py <File yang ingin dicek>”, kemudian program akan langsung mengecek file tersebut. Program memiliki keluaran antara “Accepted!!!” atau “Syntax error”. Apabila program memiliki kesulitan dalam mengubah file yang dimasukkan, program akan mengeluarkan runtime error dengan baris di mana program kesulitan mengubahnya menjadi token.

B. Pengujian

a. Hasil kasus uji inputAcc.py



```
18 file.close()
19
20 result = ""
21 line = int(1)
22 for token in tokenizer.tokenize(statements):
23     # if token.line != line:
24     #     line += 1
25     #     result += '\n'
26     result += token.type + ' '
27 result = re.sub('2', '', result)
28 # print(result)
29 files = open('tokenized.txt', 'a')
30 files.truncate(0) # removes whatever's inside
31 files.write(result)
32 files.close()
33 tknstr = cyk.makeTokenizedString()
34 checksyntax = cyk.doCYK(tknstr, 'CNF.txt')
35 if (checksyntax == True):
36     print('Program Accepted!')
37 else:
38     print('Syntax Error!')
39
40 else:
41     print('file '+sys.argv[1]+' tidak ditemukan!')
```

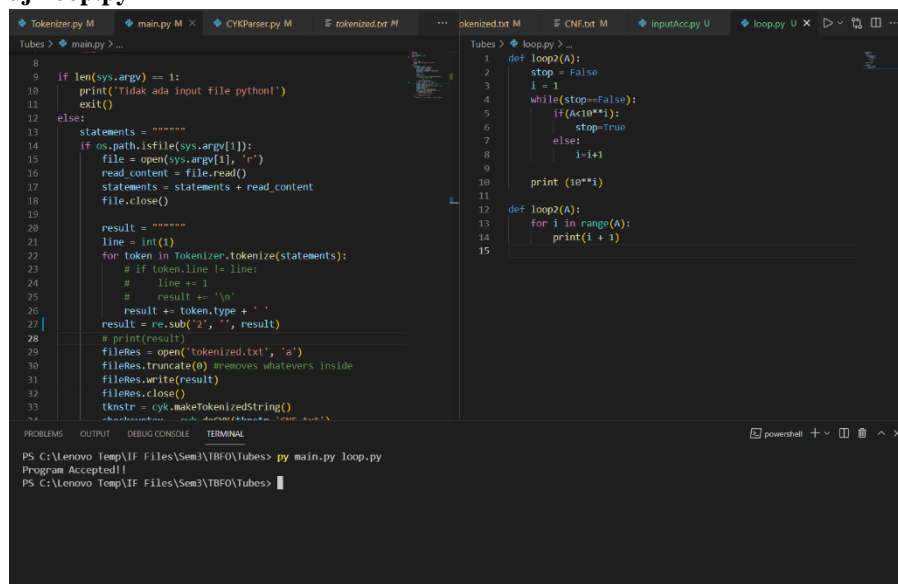
```
1 def do_something(x):
2     """ This is a sample multiline comment
3     ...
4     if x == 0:
5         return 0
6     elif x + 4 == 1:
7         if True:
8             return 3
9         else:
10            return 2
11     elif x == 32:
12         return 4
13     else:
14         return "doodoo"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

```
PS C:\lenovo Temp\IF Files\Sem3\TBFO\Tubes> py main.py inputAcc.py
Program Accepted!
PS C:\lenovo Temp\IF Files\Sem3\TBFO\Tubes>
```

Gambar 3.1 hasil uji kasus inputAcc.py

Pada kasus uji ini, kita menggunakan program untuk mengecek apakah file inputAcc.py yang kami ambil dari spesifikasi tugas besar dapat dikompilasi. Kasus ini akan mengecek komentar multibaris, if-elif-else, return, perbandingan variabel, dan seterusnya.

b. Hasil kasus uji loop.py



```
8
9 if len(sys.argv) == 1:
10     print('Tidak ada input file python!')
11     exit()
12 else:
13     statements = ""
14     if os.path.isfile(sys.argv[1]):
15         file = open(sys.argv[1], 'r')
16         read_content = file.read()
17         statements = statements + read_content
18         file.close()
19
20 result = ""
21 line = int(1)
22 for token in Tokenizer.tokenize(statements):
23     # if token.line != line:
24     #     line += 1
25     #     result += '\n'
26     result += token.type + ' '
27 result = re.sub('2', '', result)
28 # print(result)
29 files = open('tokenized.txt', 'a')
30 files.truncate(0) # removes whatever's inside
31 files.write(result)
32 files.close()
33 tknstr = cyk.makeTokenizedString()
34 checksyntax = cyk.doCYK(tknstr, 'CNF.txt')
```

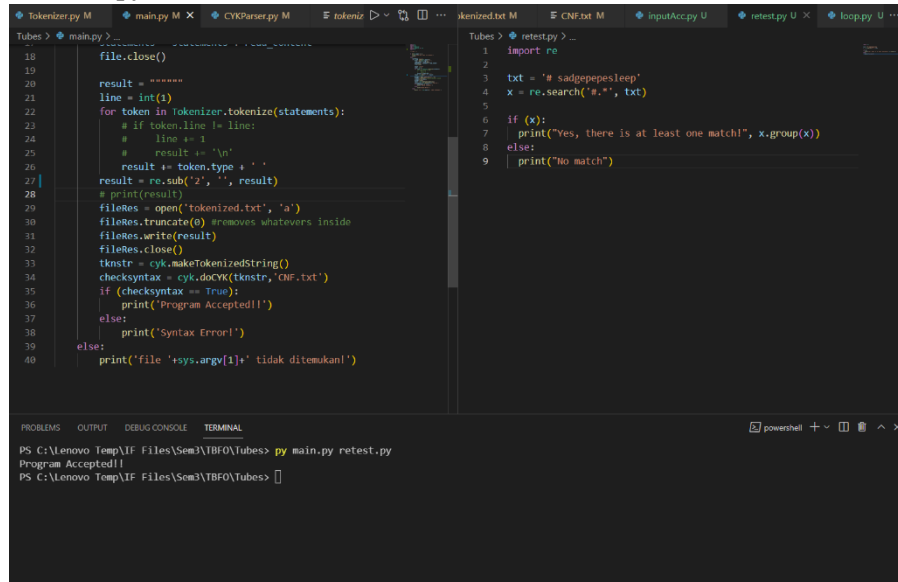
```
1 def loop2(A):
2     stop = False
3     i = 1
4     while(stop==False):
5         if(A<10**i):
6             stop=True
7         else:
8             i=i+1
9
10    print (10**i)
11
12 def loop(A):
13     for i in range(A):
14         print(i + 1)
15
```

```
PS C:\lenovo Temp\IF Files\Sem3\TBFO\Tubes> py main.py loop.py
Program Accepted!
PS C:\lenovo Temp\IF Files\Sem3\TBFO\Tubes>
```

Gambar 3.2 hasil uji kasus loop.py

Pada kasus uji ini, kita menggunakan program untuk mengecek apakah file input loop.py dapat dikompilasi. Kasus ini akan mengecek for dan while loop serta print.

c. Hasil uji kasus retest.py



```
18 file.close()
19
20 result = ""
21 line = int(1)
22 for token in tokenizer.tokenize(statements):
23     # if token.line != line:
24     #     line += 1
25     #     result += '\n'
26     result += token.type + ' '
27 result = re.sub('2', '', result)
28 # print(result)
29 files = open('tokenized.txt', 'a')
30 files.truncate(0) #removes whatever's inside
31 files.write(result)
32 files.close()
33 tknstr = cyk.makeTokenizedString()
34 checksyntax = cyk.doCVK(tknstr, 'CNF.txt')
35 if (checksyntax == True):
36     print("Program Accepted!!")
37 else:
38     print("Syntax Error!!")
39 else:
40     print('file '+sys.argv[1]+' tidak ditemukan!')
```

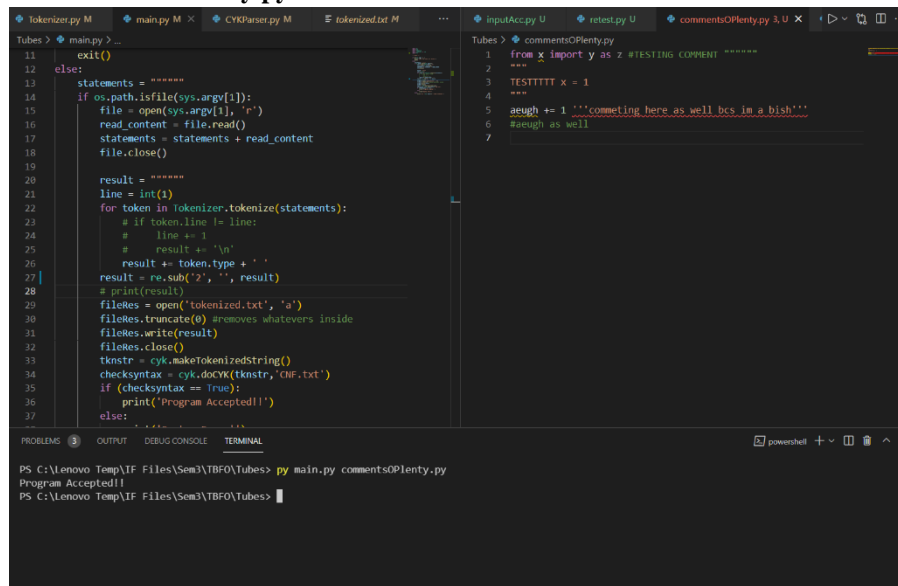
```
1 import re
2
3 txt = '# sadgepepesleep'
4 x = re.search("#.", txt)
5
6 if (x):
7     print("Yes, there is at least one match!", x.group(x))
8 else:
9     print("No match")
```

```
PS C:\Lenovo Temp\IF Files\Sem3\TBFO\Tubes> py main.py retest.py
Program Accepted!!
PS C:\Lenovo Temp\IF Files\Sem3\TBFO\Tubes>
```

Gambar 3.3 hasil uji kasus retest.py

Pada kasus uji ini, kita menggunakan program untuk mengecek apakah file input retest.py dapat dikompilasi. Kasus ini akan memiliki import suatu modul dari library python dan menggunakan modul tersebut.

d. Hasil uji kasus commentsOPlentY.py



```
11 exit()
12 else:
13     statements = ""
14     if os.path.isfile(sys.argv[1]):
15         file = open(sys.argv[1], 'r')
16         read_content = file.read()
17         statements = statements + read_content
18         file.close()
19
20 result = ""
21 line = int(1)
22 for token in tokenizer.tokenize(statements):
23     # if token.line != line:
24     #     line += 1
25     #     result += '\n'
26     result += token.type + ' '
27 result = re.sub('2', '', result)
28 # print(result)
29 files = open('tokenized.txt', 'a')
30 files.truncate(0) #removes whatever's inside
31 files.write(result)
32 files.close()
33 tknstr = cyk.makeTokenizedString()
34 checksyntax = cyk.doCVK(tknstr, 'CNF.txt')
35 if (checksyntax == True):
36     print("Program Accepted!!")
37 else:
```

```
1 from x import y as z #TESTING COMMENT *****
2
3 TESTTTTT x = 1
4
5 azeugh += 1 '''commenting here as well bcs in a bish'''
6
7 #azeugh as well
```

```
PS C:\Lenovo Temp\IF Files\Sem3\TBFO\Tubes> py main.py commentsOPlentY.py
Program Accepted!!
PS C:\Lenovo Temp\IF Files\Sem3\TBFO\Tubes>
```

Gambar 3.4 hasil uji kasus commentsOPlentY.py

Pada kasus uji ini, kita menggunakan program untuk mengecek apakah file input commentsOPlentY.py dapat dikompilasi. Kasus ini akan menggunakan semua jenis komentar python dari '#' hingga multibaris ('''''''') dan ('''''').

LINK REPOSITORY GITHUB

<https://github.com/Noxira/TBFO-CYK-Py.git>

PEMBAGIAN TUGAS

No.	NIM	Nama	Tugas
1.	13520003	Dzaky Fattan Rizqullah	Laporan, Grammar CFG, CNF
2.	13520038	Shadiq Harwiz	Laporan, testing
3.	13520054	Farrel Farandieka Fibriyanto	Laporan, tokenizer, CYK

REFERENSI

Automata CYK: mmheydari97 @github (<https://github.com/mmheydari97/automata-cyk>)

Fairuzabadi, Muhammad, Bentuk Normal Chomsky (<https://fairuzelsaid.wordpress.com/2011/06/16/tbo-context-free-grammar-cfg/>)

Fairuzabadi, Muhammad, “Bentuk Normal Chomsky” (<https://fairuzelsaid.wordpress.com/2011/06/23/bentuk-normal-chomsky/>)

Keyword Python (<https://www.programiz.com/python-programming/keyword-list>)

Regexr: Build & Test Regular Expressions (<https://regexr.com/>)

RE Documentary Python: Writing a Tokenizer (<https://docs.python.org/3/library/re.html#writing-a-tokenizer>)

Wikipedia: CYK Algorithm (https://en.wikipedia.org/wiki/CYK_algorithm)

Wikipedia: Python ([https://id.wikipedia.org/wiki/Python_\(bahasa_pemrograman\)](https://id.wikipedia.org/wiki/Python_(bahasa_pemrograman)))