

IF2124 Teori Bahasa Formal dan Otomata

Laporan Tugas Besar *Compiler* Bahasa Python



Oleh:

Kelompok YNTKTS

Mohamad Daffa Argakoesoemah 13520118

Ikmal Alfaozi 13520125

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

BAB I

TEORI DASAR

1.1 *Finite Automata (FA)*

Finite automata adalah otomata yang mempunyai himpunan *state* atau keadaan dan “kontrol” pergerakan sebagai respons atas masukan eksternal. *Finite automata* biasa digunakan sebagai model untuk perangkat lunak yang digunakan untuk sirkuit digital, *lexer analyzer* untuk *compiler*, dll. *Finite automata* dibagi menjadi dua, yaitu:

1.1.1 *Deterministic Finite Automata (DFA)*

Deterministic mempunyai arti bahwa untuk setiap input, terdapat satu *state* yang bisa dicapai saat terjadi transisi. Jumlah *state* tersebut juga tidak boleh kurang (tidak ada transisi) atau lebih. Secara formal, DFA didefinisikan sebagai lima pasang nilai, yaitu:

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

| | | |
|-----------------|---|--|
| Q | : | himpunan <i>state</i> yang terbatas |
| Σ | : | simbol masukan |
| δ | : | fungsi transisi $(q, a) \rightarrow p$ |
| $q_0 \in Q$ | : | <i>start state</i> |
| $F \subseteq Q$ | : | himpunan <i>final state</i> |

1.1.2 *Nondeterministic Finite Automata (NFA)*

Berlawanan dari *deterministic* pada DFA, *nondeterministic* mempunyai arti bahwa transisi yang dilakukan untuk setiap *input* dapat menuju lebih atau kurang dari satu *state* (tidak ada transisi). Secara formal, NFA didefinisikan sebagai lima pasang nilai, yaitu:

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

| | | |
|-----------------|---|---|
| Q | : | himpunan <i>state</i> yang terbatas |
| Σ | : | simbol masukan |
| δ | : | fungsi transisi dari $Q \times \Sigma$ menuju himpunan kuasa dari Q |
| $q_0 \in Q$ | : | <i>start state</i> |
| $F \subseteq Q$ | : | himpunan <i>final state</i> |

1.2 Context-Free Grammar (CFG)

Tata bahasa bebas konteks atau *context-free grammar* merupakan notasi formal untuk mengekspresikan definisi rekursif dari bahasa. Empat komponen penting pada CFG, yaitu:

1. Terdapat himpunan terbatas dari simbol-simbol yang membentuk *string* dalam suatu bahasa
2. Terdapat himpunan terbatas dari variabel yang disebut *nonterminals* atau *syntactic categories*.
3. Salah satu variabel yang merepresentasikan suatu bahasa disebut *start symbol*.
4. Terdapat himpunan terbatas dari produksi atau aturan yang merepresentasikan definisi rekursif dari suatu bahasa. Setiap produksi mempunyai komponen sebagai berikut:
 - a. Variabel yang didefinisikan oleh suatu produksi yang disebut *head* dari suatu produksi
 - b. Simbol produksi \rightarrow
 - c. *String* dari nol atau lebih *terminal* dan variabel. *String* ini disebut *body* dari suatu produksi

Keempat komponen di atas membentuk CFG. Secara formal CFG didefinisikan sebagai empat pasang nilai, yaitu:

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

| | | |
|---|---|--|
| V | : | himpunan terbatas variabel |
| T | : | himpunan terbatas <i>terminal</i> |
| P | : | himpunan terbatas produksi dalam bentuk $A \rightarrow \alpha$ dengan A adalah variabel dan $\alpha \in (V \cup T)^*$ |
| S | : | <i>start symbol</i> |

1.3 Chomsky Normal Form (CNF)

CNF adalah singkatan dari bentuk normal Chomsky. Sebuah CFG (tata bahasa bebas konteks) termasuk ke dalam CNF (bentuk normal Chomsky) jika semua aturan produksi memenuhi salah satu kondisi berikut:

- Simbol awal menurunkan ϵ . Sebagai contoh, $S \rightarrow \epsilon$
- Satu variabel menurunkan dua variabel. Sebagai contoh, $A \rightarrow BC$
- Satu variabel menurunkan satu terminal. Sebagai contoh $A \rightarrow a$

Contoh:

$G1 = \{S \rightarrow AB, S \rightarrow c, A \rightarrow a, B \rightarrow b\}$

$G2 = \{S \rightarrow aA, A \rightarrow a, B \rightarrow c\}$

Aturan produksi grammar $G1$ memenuhi aturan yang ditentukan untuk CNF sehingga tata bahasa $G1$ termasuk ke dalam CNF. Namun, aturan produksi grammar $G2$ tidak memenuhi aturan yang ditentukan untuk CNF karena $S \rightarrow aA$ berisi terminal diikuti oleh non-terminal sehingga tata bahasa $G2$ tidak termasuk ke dalam CNF.

Langkah-langkah konversi CFG ke CNF:

1. Jika simbol awal S muncul di beberapa sisi kanan, buat simbol awal baru S' dan produksi baru $S' \rightarrow S$.
2. Hilangkan produksi null, unit, dan *useless* dengan menggunakan penyederhanaan CFG.
3. Ganti terminal dari sisi kanan produksi jika terminal tersebut berada dengan simbol non-terminal atau terminal lainnya. Misalnya, produksi $S \rightarrow aA$ dapat didekomposisi sebagai:

$$S \rightarrow RA$$
$$R \rightarrow a$$

4. Ganti setiap produksi $A \rightarrow B1 \dots Bn$ dengan $A \rightarrow B1C$, $n > 2$ dan $C \rightarrow B2 \dots Bn$. Ulangi langkah ini untuk semua produksi yang memiliki dua atau lebih simbol di sisi kanan.

Contoh:

Konversi CFG berikut ke CNF

$$S \rightarrow a \mid aA \mid B$$
$$A \rightarrow aBB \mid \epsilon$$
$$B \rightarrow Aa \mid b$$

Langkah 1: Jika simbol awal S muncul di beberapa sisi kanan, buat simbol awal baru S' dan produksi baru $S' \rightarrow S$.

Karena tidak ada simbol S yang muncul di sisi kanan, simbol baru tidak dibuat.

Langkah 2: Hilangkan produksi null, unit, dan *useless* dengan menggunakan penyederhanaan CFG.

Grammar $G1$ mengandung produksi null $A \rightarrow \epsilon$ sehingga produksi tersebut harus dihapus. Grammar yang dihasilkan:

$$S \rightarrow a \mid aA \mid B$$
$$A \rightarrow aBB$$
$$B \rightarrow Aa \mid b \mid a$$

Grammar $G1$ mengandung produksi unit $S \rightarrow B$ sehingga produksi tersebut harus dihapus. Hasil penghapusannya:

$$S \rightarrow a \mid aA \mid Aa \mid b$$

$A \rightarrow aBB$
 $B \rightarrow Aa \mid b \mid a$

Langkah 3: Ganti terminal dari sisi kanan produksi jika terminal tersebut berada dengan simbol non-terminal atau terminal lainnya.

$S \rightarrow a \mid XA \mid AX \mid b$
 $A \rightarrow XBB$
 $B \rightarrow AX \mid b \mid a$
 $X \rightarrow a$

Langkah 4: Ganti setiap produksi $A \rightarrow B_1 \dots B_n$ dengan $A \rightarrow B_1 C$, $n > 2$ dan $C \rightarrow B_2 \dots B_n$. Ulangi langkah ini untuk semua produksi yang memiliki dua atau lebih simbol di sisi kanan.

$S \rightarrow a \mid XA \mid AX \mid b$
 $A \rightarrow YB$
 $B \rightarrow AX \mid b \mid a$
 $X \rightarrow a$
 $Y \rightarrow XB$

1.4 Cocke-Younger Kasami (CYK)

Algoritma CYK adalah sebuah algoritma yang digunakan untuk membuktikan apakah sebuah word w di-generate oleh bahasa bebas konteks (CFG) atau tidak. Algoritma CYK dikembangkan oleh John Cocke, Daniel Younger, dan Tadao Kasami. Untuk dapat menggunakan algoritma ini dibutuhkan CFG G yang sudah dimodifikasi ke dalam bentuk CNF, dengan word w adalah sebagai input, dan outputnya adalah sebuah pembuktian apakah word w merupakan bahasa dari grammar G atau bukan.

Kita dapat mengetahui variabel apa saja yang dapat menurunkan suatu string w , $w \in \Sigma^*$, dengan menggunakan algoritma CYK ini. Ada dua kemungkinan suatu string dapat diturunkan, yaitu:

- Kemungkinan 1: $w = a \in \Sigma$, x terdiri dari simbol alphabet tunggal. Kemudian w hanya bisa diturunkan dari variabel A jika terdapat $A \rightarrow a$.
- Kemungkinan 2: $w = a_1 a_2 \dots a_n$ dengan $n \geq 2$. Pada kasus ini produksi $A \rightarrow BC$ harus dipilih terlebih dahulu sehingga satu bagian dari string $a_1 \dots a_k$ harus diturunkan dari B dan satu bagian lagi dari string $a_{k+1} \dots a_n$ diturunkan dari C ($1 \leq k < n$).

Namun, pada kemungkinan kedua kita akan mengalami kesulitan untuk memisahkan string w menjadi dua bagian karena indeks k yang besar. Kita harus mencoba semua kemungkinan k . Jika diketahui $w = a_1 a_2 \dots a_n$, dengan $1 < k < n$ lakukan langkah berikut:

- Cek apakah himpunan variabel $V1$ dapat menurunkan $a_1 \dots a_k$.
- Cek apakah himpunan variabel $V2$ dapat menurunkan $a_{k+1} \dots a_n$.
- Cek apakah variabel A, B, C dimana $(A \rightarrow BC) \in P, B \in V1$ dan $C \in V2$.
- w dapat diturunkan dari A jika semua kondisi di atas terpenuhi.

Untuk menghindari duplikasi, aplikasikan metode *dynamic programming*, yaitu:

- Tentukan dahulu semua variabel yang bisa menurunkan substring dengan panjang 1.
- Kemudian tentukan variabel yang bisa menurunkan substring dengan panjang 2.
- ...
- Terakhir, tentukan semua variabel sehingga w dapat diturunkan. w adalah bahasa dari grammar tersebut jika simbol awal S berada di antara variabel.

Pada aplikasinya, kita bisa menggunakan tabel segitiga sebagai berikut:

| | | | | |
|-----------|-----------|-----------|-----------|-----------|
| $X_{1,5}$ | | | | |
| $X_{1,4}$ | $X_{2,5}$ | | | |
| $X_{1,3}$ | $X_{2,4}$ | $X_{3,5}$ | | |
| $X_{1,2}$ | $X_{2,3}$ | $X_{3,4}$ | $X_{4,5}$ | |
| $X_{1,1}$ | $X_{2,2}$ | $X_{3,3}$ | $X_{4,4}$ | $X_{5,5}$ |
| w_1 | w_2 | w_3 | w_4 | w_5 |

Gambar 1.1 Tabel segitiga CYK

- Setiap baris berhubungan dengan panjang dari substring.
 - Baris paling bawah untuk panjang substring satu.
 - Baris kedua paling bawah untuk panjang substring dua.
 - ...
 - Baris paling atas untuk panjang string w
- $X_{i,i}$ adalah himpunan variabel A sehingga $A \rightarrow w_i$ adalah produksi dari G .
- Bandingkan n pasangan dari set yang sebelumnya sudah diproses.
 $(X_{i,i}, X_{i+1,j}), (X_{i,i+1}, X_{i+2,j}) \dots (X_{i,j-1}, X_{j,j})$.

1.5 Bahasa dan Sintaks Pemrograman Python

Python adalah bahasa pemrograman *interpreter* tingkat tinggi (*high-level-language*) yang multiparadigma karena mendukung implementasi paradigma fungsional, imperatif, berorientasi objek, dan reflektif. Sintaks Python yang sederhana dan mudah dipelajari menekankan pada *readability* (keterbacaan) sehingga mengurangi biaya *program maintenance*. Python mendukung pembuatan modul dan *packages* yang mendorong modularitas program dan penggunaan kembali kode program (*code reuse*).

Pada pembuatan program dibutuhkan pemeriksaan sintaks dengan tujuan agar intruksi-intruksi yang dibuat dalam program sesuai dengan aturan bahasa pemrograman tersebut. Pemeriksaan sintaks tersebut membutuhkan *grammar* dan algoritma *parser* untuk menghasilkan *compiler* dengan performa tinggi. Banyak *grammar* dan algoritma *parser* yang sudah dikembangkan untuk pembuatan *compiler* tersebut. Misalnya untuk *grammar* sendiri sudah ada CFG, CNF^e , CNF^{+e} , 2NF, dan 2LF, sedangkan untuk algoritma *parser* terdapat LL(0), LL(1), CYK, Algoritma Earley, LALR, GLR, Shift-reduce, SLR, dan LR(1). Pada pembuatan *compiler* kali ini, kita menggunakan CFG untuk *grammar*-nya dan CYK untuk algoritma *parser*-nya.

Berikut adalah sintaks python yang kami implementasikan dalam pembuatan CFG dan FA:

Tabel 1.1 Daftar sintaks Python yang diimplementasikan

| | | | | |
|-------|----------|--------|------|--------|
| False | break | else | in | raise |
| None | class | for | is | return |
| True | continue | from | not | while |
| and | def | if | or | with |
| as | elif | import | pass | |

Masing-masing kata kunci memiliki aturan tersendiri. Misalnya untuk kata kunci “elif” dan “else”, hanya bisa digunakan setelah kata kunci “if” untuk indentasi yang sama. Kata kunci lainnya, seperti “while”, “for”, “class”, dan “def” harus diikuti baris baru untuk sintaks yang tidak bisa ditulis dalam satu baris, seperti “while”, “for”, “class”, dan “def” itu sendiri. Sementara itu, terdapat kata kunci yang harus ditulis dalam satu baris seperti “import”, “return”, “with”, dan “raise”. Keterangan lebih lanjut mengenai implementasi sintaks Python dalam CFG dapat dilihat pada bagian 2.2.

BAB II

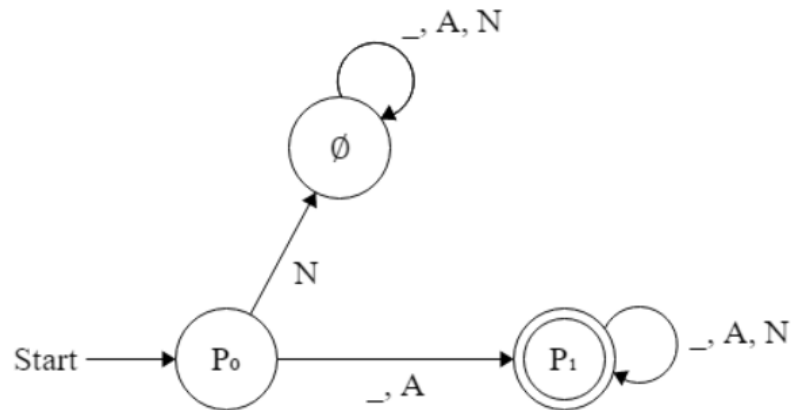
HASIL

2.3 *Deterministic Finite Automata (DFA)*

DFA digunakan dalam program ini untuk mengecek kevalidan nama variabel yang dipakai. Beberapa aturan penamaan variabel dalam bahasa Python adalah:

1. Nama variabel harus dimulai dengan huruf alfabet atau karakter *underscore*
2. Nama variabel tidak bisa dimulai dengan angka
3. Nama variabel hanya boleh mengandung karakter *alpha-numeric* dan karakter *underscore*
4. Nama variabel *case-sensitive*
5. *Reserved keywords* tidak bisa digunakan sebagai nama variabel, seperti if, else, while, dll.

DFA yang dibuat mengimplementasikan aturan pertama sampai ketiga. Diagram transisinya ditunjukkan oleh Gambar 2.1, sedangkan tabel transisinya ditunjukkan oleh Tabel 2.1.



Gambar 2.1 DFA untuk pengecekan nama variabel

Keterangan Gambar 1:

$A = \{A...Z, a...z\}$

$N = \{0..9\}$

Tabel 2.1 Tabel transisi DFA untuk pengecekan nama variabel

| δ | A | N | – |
|-------------------|-------------|-------------|-------------|
| $\rightarrow P_0$ | P_1 | \emptyset | P_1 |
| $* P_1$ | P_1 | P_1 | P_1 |
| \emptyset | \emptyset | \emptyset | \emptyset |

2.2 Context-Free Grammar (CFG)

Pada program ini, CFG digunakan untuk mengevaluasi *syntax* program. Secara formal didefinisikan sebagai:

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

Variabel (V) ditunjukkan oleh Tabel 2.2

Tabel 2.2 Variabel CFG

| | | | |
|-----------------|---------------|--------------|------------------|
| VARIABLE | ARITHMETIC_OP | WITH | LEFT_RB |
| MULTI_VAR | BREAK | AS | RIGHT_RB |
| MULTI_TYPE | CONTINUE | NONE | LEFT_SB |
| RELATIONAL_OP | PASS | INTEGER | RIGHT_SB |
| LOGICAL_OP | IN | FLOAT | COLON |
| COMMA | BOOLEAN | PRINT | COMMENT |
| PERIOD | EXPRESSION | MODULE | DATA_TYPE |
| NEWLINE | PARAM | TUPLE | ASSIGN |
| TYPE | CASTING | LIST | SELF_ARITH_RIGHT |
| BOOLEAN_OPERAND | INPUT | ARITH | SELF_ARITH |
| VALUE | WHILE_BLOCK | DEF_ONE_LINE | IMPORT |
| IF_ONE_LINE | ONE_LINE | DEF_BLOCK | FROM |
| IF_BLOCK | RANGE_TYPE | RETURN | RAISE |

| | | | |
|------------|-------------|----------------|----------------|
| ELIF_BLOCK | RANGE_PARAM | CLASS_ONE_LINE | WITH |
| ELSE_BLOCK | FOR_BLOCK | CLASS | BREAK_CONTINUE |
| | | | |

Simbol terminal (T) ditunjukkan oleh Tabel 2.3.

Tabel 2.3 Simbol terminal CFG

| | | | | |
|----------|--------|---------|----------|---------|
| variable | is | - | continue | None |
| = | and | * | pass | integer |
| ! | or | / | in | (|
| > | not | % | with |) |
| < | + | break | as | [|
|] | string | float | elif | def |
| : | True | input | else | return |
| , | False | print | while | class |
| . | int | comment | range | import |
| newline | str | if | for | from |
| raise | with | | | |

Produksi dan hasil produksi dapat dilihat pada Tabel 2.4.

Tabel 2.4 Aturan produksi CFG

| Produksi | Hasil Produksi |
|----------|--|
| S | S S S NEWLINE S NEWLINE COMMENT ASSIGN SELF_ARITH IF_BLOCK IF_ONE_LINE WHILE_BLOCK FOR_BLOCK DEF_BLOCK METHOD RETURN CLASS CLASS_OBJECT IMPORT FROM RAISE WITH PRINT INPUT PASS VALUE ONE_LINE |
| VARIABLE | variable variable PERIOD VARIABLE |

| | |
|-----------------|---|
| MULTI_CLASS | VALUE VALUE COMMA MULTI_CLASS |
| MULTI_VAR | VARIABLE VARIABLE COMMA MULTI_VAR |
| MULTI_TYPE | TYPE TYPE COMMA MULTI_TYPE LIST LIST COMMA MULTI_TYPE |
| RELATIONAL_OP | = != > < >= <= is is not |
| LOGICAL_OP | and or not |
| ARITHMETIC_OP | + - * / % ** // |
| PASS | pass |
| IN | in |
| WITH | with |
| AS | as |
| NONE | None |
| INTEGER | integer - integer |
| FLOAT | integer PERIOD integer - integer PERIOD integer |
| LEFT_RB | (|
| RUGHT_RB |) |
| LEFT_SB | [|
| RIGHT_SB |] |
| COLON | : |
| COMMA | , |
| PERIOD | . |
| NEWLINE | newline |
| ONE_LINE | VALUE ASSIGN SELF_ARITH RAISE PASS ARR_ELMT |
| TYPE | string INTEGER FLOAT |
| BOOLEAN_OPERAND | VARIABLE TYPE ARITH True False |

| | |
|---------------------|--|
| | NONE ARR_ELMT |
| OR_AND | or and |
| BOOLEAN | LEFT_RB BOOLEAN RIGHT_RB BOOLEAN_OPERAND LEFT_RB BOOLEAN_OPERAND RIGHT_RB not BOOLEAN not LEFT_RB BOOLEAN RIGHT_RB BOOLEAN_OPERAND RELATIONAL_OP BOOLEAN_OPERAND BOOLEAN OR_AND BOOLEAN LEFT_RB BOOLEAN RIGHT_RB OR_AND LEFT_RB BOOLEAN RIGHT_RB |
| EXPRESSION_ONE_LINE | BOOLEAN COLON ONE_LINE LEFT_RB BOOLEAN RIGHT_RB COLON ONE_LINE |
| EXPRESSION | BOOLEAN COLON NEWLINE S LEFT_RB BOOLEAN RIGHT_RB COLON NEWLINE S BOOLEAN COLON ONE_LINE LEFT_RB BOOLEAN RIGHT_RB COLON ONE_LINE |
| EXPRESSION_COND_BC | EXPRESSION BOOLEAN COLON NEWLINE LEFT_RB BOOLEAN RIGHT_RB COLON NEWLINE EXPRESSION_BC |
| PARAM | LEFT_RB MULTI_VAR RIGHT_RB LEFT_RB RIGHT_RB |
| CASTING | int LEFT_RB string RIGHT_RB str LEFT_RB INTEGER RIGHT_RB int LEFT_RB FLOAT RIGHT_RB float LEFT_RB INTEGER RIGHT_RB str LEFT_RB FLOAT RIGHT_RB float LEFT_RB string RIGHT_RB |
| INPUT | input LEFT_RB string RIGHT_RB |
| PRINT | print LEFT_RB VARIABLE RIGHT_RB print LEFT_RB TYPE RIGHT_RB print LEFT_RB BOOLEAN RIGHT_RB |
| MODULE | VARIABLE PERIOD VARIABLE VARIABLE MODULE |
| TUPLE | LEFT_RB MULTI_TYPE RIGHT_RB |
| LIST | LEFT_SB MULTI_TYPE RIGHT_SB LEFT_SB RIGHT_SB |

| | |
|------------------|---|
| ARITH_OPERAND | INTEGER VARIABLE FLOAT ARR_ELMT |
| ARITH | ARITH_OPERAND ARITH ARITHMETIC_OP ARITH_OPERAND |
| ARR_ELMT | VARIABLE LEFT_SB VARIABLE RIGHT_SB VARIABLE LEFT_SB INTEGER RIGHT_SB VARIABLE LEFT_SB ARITH RIGHT_SB |
| BREAK_CONTINUE | break continue S NEWLINE break S NEWLINE continue break NEWLINE S continue NEWLINE S NEWLINE break NEWLINE S NEWLINE continue NEWLINE S |
| COMMENT | comment NEWLINE |
| DATA_TYPE | TYPE TUPLE LIST |
| ASSIGN_DEST | VARIABLE ARR_ELMT CLASS_OBJECT |
| ASSIGN | ASSIGN_DEST = ARITH ASSIGN_DEST = DATA_TYPE ASSIGN_DEST = VARIABLE ASSIGN_DEST = ASSIGN ASSIGN_DEST = CLASS_OBJECT |
| SELF_ARITH_RIGHT | INTEGER VARIABLE FLOAT ARR_ELMT ARITH |
| SELF_ARITH | SELF_ARITH -> ASSIGN_DEST + = SELF_ARITH_RIGHT ASSIGN_DEST + = string ASSIGN_DEST - = SELF_ARITH_RIGHT ASSIGN_DEST * = SELF_ARITH_RIGHT ASSIGN_DEST / = SELF_ARITH_RIGHT ASSIGN_DEST % = SELF_ARITH_RIGHT ASSIGN_DEST // = SELF_ARITH_RIGHT ASSIGN_DEST ** = SELF_ARITH_RIGHT |
| VALUE | TYPE TUPLE LIST VARIABLE ARR_ELMT |
| IF_ONE_LINE | VALUE if BOOLEAN else ONE_LINE VARIABLE = VALUE if BOOLEAN else ONE_LINE |
| IF_BLOCK | if EXPRESSION if EXPRESSION ELIF_BLOCK if EXPRESSION else COLON NEWLINE S if EXPRESSION else COLON |

| | |
|----------------|--|
| | ONE_LINE |
| ELIF_BLOCK | elif EXPRESSION elif EXPRESSION ELIF_BLOCK elif EXPRESSION else COLON NEWLINE S elif EXPRESSION else COLON ONE_LINE |
| ELSE_BLOCK | else COLON NEWLINE S else COLON ONE_LINE |
| IF_BLOCK_COND | if EXPRESSION_BC if EXPRESSION BREAK_CONTINUE IF_BLOCK NEWLINE IF_BLOCK_BC IF_BLOCK ELIF_BLOCK_BC IF_BLOCK ELSE_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK IF_BLOCK NEWLINE IF_BLOCK_BC IF_BLOCK ELIF_BLOCK_BC IF_BLOCK ELSE_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK |
| EXPRESSION_BC | EXPRESSION BOOLEAN COLON NEWLINE BREAK_CONTINUE NEWLINE LEFT_RB BOOLEAN RIGHT_RB COLON NEWLINE BREAK_CONTINUE NEWLINE BOOLEAN COLON BREAK_CONTINUE NEWLINE LEFT_RB BOOLEAN RIGHT_RB COLON BREAK_CONTINUE NEWLINE EXPRESSION BOOLEAN COLON NEWLINE BREAK_CONTINUE NEWLINE LEFT_RB BOOLEAN RIGHT_RB COLON NEWLINE BREAK_CONTINUE NEWLINE BOOLEAN COLON BREAK_CONTINUE NEWLINE LEFT_RB BOOLEAN RIGHT_RB COLON BREAK_CONTINUE NEWLINE |
| IF_ONE_LINE_BC | VALUE if BOOLEAN else break VALUE if BOOLEAN else continue |
| IF_BLOCK_BC | if EXPRESSION_COND_BC IF_BLOCK_BC IF_BLOCK S BREAK_CONTINUE S if EXPRESSION BREAK_CONTINUE if EXPRESSION NEWLINE IF_BLOCK_BC if EXPRESSION_BC IF_BLOCK_BC ELIF_BLOCK_BC IF_BLOCK_BC ELSE_BLOCK_BC IF_BLOCK_BC ELIF_BLOCK IF_BLOCK_BC ELSE_BLOCK IF_BLOCK NEWLINE |

| | |
|---------------|--|
| | IF_BLOCK_BC IF_BLOCK ELIF_BLOCK_BC IF_BLOCK ELSE_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK_BC IF_BLOCK |
| ELIF_BLOCK_BC | elif EXPRESSION BREAK_CONTINUE elif EXPRESSION_BC ELIF_BLOCK_BC ELIF_BLOCK_BC ELIF_BLOCK_BC ELSE_BLOCK_BC ELIF_BLOCK_BC ELIF_BLOCK ELIF_BLOCK_BC ELSE_BLOCK ELIF_BLOCK ELIF_BLOCK_BC ELIF_BLOCK ELSE_BLOCK_BC ELIF_BLOCK_BC IF_BLOCK_BC ELIF_BLOCK_BC IF_BLOCK ELIF_BLOCK IF_BLOCK_BC ELIF_BLOCK IF_BLOCK_BC |
| ELSE_BLOCK_BC | else COLON NEWLINE BREAK_CONTINUE else COLON BREAK_CONTINUE else COLON NEWLINE S BREAK_CONTINUE |
| WHILE_BLOCK | while EXPRESSION while EXPRESSION_ONE_LINE while EXPRESSION BREAK_CONTINUE while EXPRESSION_BC NEWLINE S while EXPRESSION IF_BLOCK_BC while EXPRESSION IF_BLOCK_BC NEWLINE S while EXPRESSION IF_ONE_LINE_BC NEWLINE S while EXPRESSION IF_BLOCK_BC NEWLINE IF_BLOCK while EXPRESSION NEWLINE IF_BLOCK_COND while EXPRESSION_COND_BC IF_BLOCK_BC while EXPRESSION_COND_BC IF_BLOCK_BC NEWLINE S while EXPRESSION_COND_BC IF_BLOCK_BC NEWLINE IF_BLOCK while EXPRESSION_COND_BC IF_BLOCK NEWLINE IF_BLOCK_BC |
| RANGE_TYPE | INTEGER VARIABLE ARITH |
| RANGE_PARAM | range LEFT_RB RANGE_TYPE RIGHT_RB range LEFT_RB RANGE_TYPE COMMA RANGE_TYPE RIGHT_RB range LEFT_RB RANGE_TYPE COMMA RANGE_TYPE COMMA RANGE_TYPE RIGHT_RB |

| | |
|--------------|---|
| FOR_PARAM | VARIABLE IN VALUE VARIABLE IN RANGE_PARAM |
| FOR_BLOCK | for FOR_PARAM COLON ONE_LINE for FOR_PARAM COLON NEWLINE S for FOR_PARAM COLON NEWLINE BREAK_CONTINUE for FOR_PARAM COLON NEWLINE IF_BLOCK_BC for FOR_PARAM COLON NEWLINE IF_BLOCK_BC NEWLINE S for FOR_PARAM COLON NEWLINE IF_BLOCK_BC NEWLINE S for FOR_PARAM COLON NEWLINE IF_ONE_LINE_BC NEWLINE S for FOR_PARAM COLON NEWLINE IF_BLOCK_BC NEWLINE IF_BLOCK for FOR_PARAM COLON NEWLINE IF_BLOCK_COND for FOR_PARAM COLON NEWLINE IF_BLOCK NEWLINE IF_BLOCK_BC for FOR_PARAM COLON NEWLINE IF_BLOCK_BC NEWLINE IF_BLOCK |
| METHOD_PARAM | VALUE ASSIGN |
| METHOD | VARIABLE PERIOD LEFT_RB METHOD_PARAM RIGHT_RB |
| DEF_BLOCK | def VARIABLE PARAM COLON ONE_LINE def VARIABLE PARAM COLON NEWLINE S |
| RETURN | return BOOLEAN return VARIABLE return DATA_TYPE |
| PARAM_CLASS | LEFT_RB MULTI_CLASS RIGHT_RB LEFT_RB RIGHT_RB |
| CLASS | class VARIABLE COLON ONE_LINE class VARIABLE COLON NEWLINE S |
| CLASS_OBJECT | VARIABLE PARAM_CLASS |
| IMPORT | import MODULE IMPORT MODULE AS VARIABLE |
| FROM | from MODULE IMPORT |
| RAISE | raise RAISE VARIABLE RAISE VARIABLE |

| | |
|------|--|
| | LEFT_RB string RIGHT_RB RAISE VARIABLE LEFT_RB string RIGHT_RB from VARIABLE RAISE VARIABLE LEFT_RB string RIGHT_RB from NONE |
| WITH | VARIABLE PARAM AS VARIABLE COLON NEWLINE S |

Berikut ini adalah hasil konversi dari CFG ke CNF dalam bentuk *dictionary* di python dengan *key* sebagai variabel dan *value* sebagai hasil produksinya.

```
[['S': [['S', 'S'], ['S1', 'S'], ['NEWLINE'], ['COMMENT'], ['ASSIGN'], ['SELF_ARITH'], ['IF_BLOCK'], ['IF_ONE_LINE'], ['WHILE_BLOCK'], ['FOR_BLOCK'], ['DEF_BLOCK'], ['METHOD'], ['RETURN'], ['CLASS'], ['CLASS_OBJECT'], ['IMPORT'], ['FROM'], ['RAISE'], ['WITH'], ['PRINT'], ['INPUT'], ['PASS'], ['VALUE'], ['ONE_LINE']], 'VARIABLE': [['variable'], ['VARIABLE1', 'VARIABLE'], ['MULTI_CLASS': ['VALUE'], ['MULTI_CLASS1', 'MULTI_CLASS'], ['MULTI_VAR': [['VARIABLE'], ['MULTI_VAR1', 'MULTI_VAR']], 'MULTI_TYPE': [['TYPE'], ['MULTI_TYPE1', 'MULTI_TYPE'], ['LIST'], ['MULTI_TYPE2', 'MULTI_TYPE']], 'RELATIONAL_OP': [['T1', 'T1'], ['T2', 'T1'], ['>'], ['<'], ['T21', 'T1'], ['T22', 'T1'], ['is'], ['T23', 'T24'], 'LOGICAL_OP': [['and'], ['or'], ['not'], 'ARITHMETIC_OP': [['+'], ['-'], ['*'], ['/'], ['%'], ['T2', 'T25'], ['T26', 'T26']], 'PASS': [['pass'], 'IN': [['in'], 'WITH': [['with'], ['WITH5', 'T3'], 'AS': [['as'], 'NONE': [['None'], 'INTEGER': [['integer'], ['T27', 'T28']], 'FLOAT': [['FLOAT1', 'T28'], ['FLOAT3', 'T28']], 'LEFT_RB': [['('], 'RIGHT_RB': [')']], 'LEFT_SB': [['['], 'RIGHT_SB': [']']], 'COLON': [[':'], 'COMMA': [[','], 'PERIOD': [['.'], 'NEWLINE': ['newline'], 'ONE_LINE': [['VALUE'], ['ASSIGN'], ['SELF_ARITH'], ['RAISE'], ['PASS'], ['ARR_ELMT'], 'TYPE': [['string'], 'INTEGER'], ['FLOAT'], 'BOOLEAN_OP': [['VARIABLE'], ['TYPE'], ['ARITH'], ['True'], ['False'], ['NONE'], ['ARR_ELMT'], 'OR_AND': [['or'], ['and']], 'BOOLEAN': [['BOOLEAN1', 'RIGHT_RB'], ['BOOLEAN_OPERAND'], ['BOOLEAN2', 'RIGHT_RB'], ['T24', 'BOOLEAN'], ['BOOLEAN4', 'RIGHT_RB'], ['BOOLEANS', 'BOOLEAN_OPERAND'], ['BOOLEAN6', 'BOOLEAN'], ['BOOLEAN11', 'RIGHT_RB']], 'EXPRESSION_ONE_LINE': [['EXPRESSION_ONE_LINE1', 'ONE_LINE'], ['EXPRESSION_ONE_LINE4', 'ONE_LINE'], 'EXPRESSION': [['EXPRESSION2', 'S'], ['EXPRESSION6', 'S'], ['EXPRESSION7', 'ONE_LINE'], ['EXPRESSION10', 'ONE_LINE'], 'EXPRESSION_COND_BC': [['EXPRESSION'], ['EXPRESSION_COND_BC1', 'NEWLINE'], ['EXPRESSION_COND_BC4', 'NEWLINE'], ['EXPRESSION_BC'], 'PARAM': [['PARAM1', 'RIGHT_RB'], ['LEFT_RB', 'RIGHT_RB'], 'CASTING': [['CASTING2', 'RIGHT_RB'], ['CASTING4', 'RIGHT_RB'], ['CASTING6', 'RIGHT_RB'], ['CASTING8', 'RIGHT_RB'], ['CASTING10', 'RIGHT_RB'], ['CASTING12', 'RIGHT_RB'], 'INPUT': [['INPUT2', 'RIGHT_RB'], 'PRINT': [['PRINT2', 'RIGHT_RB'], ['PRINT4', 'RIGHT_RB'], ['PRINT6', 'RIGHT_RB'], 'MODULE': [['VARIABLE'], ['PERIOD', 'VARIABLE'], ['VARIABLE', 'MODULE']], 'TUPLE': [['TUPLE1', 'RIGHT_RB'], 'LIST': [['LIST1', 'RIGHT_SB'], ['LEFT_SB', 'RIGHT_SB']], 'ARITH_OPERAND': [['INTEGER'], ['VARIABLE'], ['FLOAT'], ['ARR_ELMT']], 'ARITH': [['ARITH_OPERAND'], ['ARITH1', 'ARITH_OPERAND'], 'ARR_ELMT': [['ARR_ELMT2', 'RIGHT_SB'], ['ARR_ELMT4', 'RIGHT_SB'], ['ARR_ELMT6', 'RIGHT_SB'], 'BREAK_CONTINUE': [['break'], ['continue'], ['BREAK_CONTINUE1', 'T30'], ['BREAK_CONTINUE2', 'T31'], ['BREAK_CONTINUE3', 'S'], ['BREAK_CONTINUE4', 'S'], ['BREAK_CONTINUE6', 'S'], ['BREAK_CONTINUE8', 'S']], 'COMMENT': [['T8', 'NEWLINE'], 'DATA_TYPE': [['TYPE'], ['TUPLE'], ['LIST'], 'ASSIGN_DEST': [['VARIABLE'], ['ARR_ELMT'], ['CLASS_OBJECT']], 'ASSIGN': [['ASSIGN1', 'ARITH'], ['ASSIGN2', 'DATA_TYPE'], ['ASSIGN3', 'VARIABLE'], ['ASSIGN4', 'AS_SIGN'], ['ASSIGNS', 'CLASS_OBJECT']], 'SELF_ARITH_RIGHT': [['INTEGER'], ['VARIABLE'], ['FLOAT'], ['ARR_ELMT'], ['ARITH']], 'SELF_ARITH': [['SELF_ARITH2', 'SELF_ARITH_RIGHT'], ['SELF_ARITH4', 'T29'], ['SELF_ARITH6', 'SELF_ARITH_RIGHT'], ['SELF_ARITH8', 'SELF_ARITH_RIGHT'], ['SELF_ARITH10', 'SELF_ARITH_RIGHT'], ['SELF_ARITH12', 'SELF_ARITH_RIGHT'], ['SELF_ARITH15', 'SELF_ARITH_RIGHT'], ['SELF_ARITH18', 'SELF_ARITH_RIGHT'], 'VALUE': [['TYPE'], ['TUPLE'], ['LIST'], ['VARIABLE'], ['ARR_ELMT']], 'IF_ONE_LINE': [['IF_ONE_LINE3', 'ONE_LINE'], ['IF_ONE_LINE8', 'ONE_LINE'], 'IF_BLOCK': [['T9', 'EXPRESSION'], ['IF_BLOCK1', 'ELIF_BLOCK'], ['IF_BLOCKS', 'S'], ['IF_BLOCK8', 'ONE_LINE'], 'ELIF_BLOCK': [['T11', 'EXPRESSION'], ['ELIF_BLOCK1', 'ELIF_BLOCK'], ['ELIF_BLOCKS', 'S'], ['ELIF_BLOCK8', 'ONE_LINE'], 'ELSE_BLOCK': [['ELSE_BLOCK2', 'S'], ['ELSE_BLOCK3', 'ONE_LINE'], 'IF_BLOCK_COND': [['T9', 'EXPRESSION_BC'], ['IF_BLOCK_COND1', 'BREAK_CONTINUE'], ['IF_BLOCK_COND2', 'IF_BLOCK_BC'], ['IF_BLOCK', 'ELIF_BLOCK_BC'], ['IF_BLOCK', 'ELSE_BLOCK_BC'], ['IF_BLOCK_BC', 'IF_BLOCK_BC'], ['IF_BLOCK_BC', 'IF_BLOCK'], ['IF_BLOCK_COND3', 'IF_BLOCK_BC'], ['IF_BLOCK_COND', 'ELIF_BLOCK_BC'], ['IF_BLOCK_COND', 'ELSE_BLOCK_BC'], ['IF_BLOCK_BC', 'IF_BLOCK_BC'], ['IF_BLOCK_BC', 'IF_BLOCK_COND'], 'EXPRESSION_BC': [['EXPRESSION'], ['EXPRESSION_BC3', 'NEWLINE'], ['EXPRESSION_BC8', 'NEWLINE'], ['EXPRESSION_BC10', 'NEWLINE'], ['EXPRESSION_BC14', 'NEWLINE'], ['EXPRESSION_BC16', 'BREAK_CONTINUE'], ['EXPRESSION_BC20', 'BREAK_CONTINUE'], ['EXPRESSION_BC21', 'BREAK_CONTINUE'], ['EXPRESSION_BC24', 'BREAK_CONTINUE'], 'IF_
```

```

ONE_LINE_BC': [['IF_ONE_LINE_BC3', 'T30'], ['IF_ONE_LINE_BC6', 'T31']], 'IF_BLOCK_BC': [['IF_BLOCK_BC1', 'IF_BLOCK_BC'], ['IF_BLOCK_BC3', 'S'], ['IF_BLOCK_BC4', 'BREAK_CONTINUE'], ['IF_BLOCK_BC6', 'IF_BLOCK_BC'], ['T9', 'EXPRESSION_BC'], ['IF_BLOCK_BC', 'ELIF_BLOCK_BC3'], ['IF_BLOCK_BC', 'ELSE_BLOCK_BC'], ['IF_BLOCK_BC', 'ELIF_BLOCK'], ['IF_BLOCK_BC', 'ELSE_BLOCK'], ['IF_BLOCK_BC7', 'IF_BLOCK_BC'], ['IF_BLOCK', 'ELIF_BLOCK_BC'], ['IF_BLOCK', 'ELSE_BLOCK_BC'], ['IF_BLOCK_BC', 'IF_BLOCK_BC'], ['IF_BLOCK_BC'], ['IF_BLOCK'], ['ELIF_BLOCK_BC'], ['ELIF_BLOCK_BC1', 'BREAK_CONTINUE'], ['T11', 'EXPRESSION_BC'], ['ELIF_BLOCK_BC', 'ELIF_BLOCK_BC'], ['ELIF_BLOCK_BC', 'ELSE_BLOCK_BC'], ['ELIF_BLOCK_BC', 'ELIF_BLOCK'], ['ELIF_BLOCK_BC', 'ELSE_BLOCK'], ['ELIF_BLOCK', 'ELIF_BLOCK_BC'], ['ELIF_BLOCK', 'ELSE_BLOCK_BC'], ['ELIF_BLOCK_BC', 'IF_BLOCK_BC'], ['ELIF_BLOCK_BC', 'IF_BLOCK'], ['ELIF_BLOCK', 'IF_BLOCK_BC'], ['ELIF_BLOCK', 'IF_BLOCK_BC'], ['ELSE_BLOCK_BC', 'BREAK_CONTINUE'], ['ELSE_BLOCK_BC3', 'BREAK_CONTINUE'], ['ELSE_BLOCK_BC6', 'BREAK_CONTINUE']], 'WHILE_BLOCK': [['T12', 'EXPRESSION'], ['WHILE_BLOCK2', 'BREAK_CONTINUE'], ['T12', 'EXPRESSION_ONE_LINE'], ['WHILE_BLOCK3', 'BREAK_CONTINUE'], ['WHILE_BLOCK5', 'S'], ['WHILE_BLOCK6', 'IF_BLOCK_BC'], ['WHILE_BLOCK9', 'S'], ['WHILE_BLOCK12', 'S'], ['WHILE_BLOCK15', 'S'], ['WHILE_BLOCK18', 'IF_BLOCK'], ['WHILE_BLOCK20', 'IF_BLOCK_COND'], ['WHILE_BLOCK21', 'IF_BLOCK_BC'], ['WHILE_BLOCK24', 'S'], ['WHILE_BLOCK27', 'IF_BLOCK'], ['WHILE_BLOCK30', 'IF_BLOCK_BC'], ['WHILE_BLOCK31', 'IF_BLOCK_BC']], 'FOR_ONE_LINE': [['VALUE'], ['ASSIGN'], ['SELF_ARITH', 'RAISE']], 'RANGE_TYPE': [['INTEGER'], ['VARIABLE'], ['ARITH']], 'RANGE_PARAM': [['RANGE_PARAM2', 'RIGHT_RB'], ['RANGE_PARAM6', 'RIGHT_RB'], ['RANGE_PARAM12', 'RIGHT_RB']], 'FOR_PARAM': [['FOR_PARAM1', 'VALUE'], ['FOR_PARAM2', 'RANGE_PARAM'], ['FOR_BLOCK', 'FOR_BLOCK2', 'ONE_LINE'], ['FOR_BLOCK5', 'S'], ['FOR_BLOCK8', 'EXPRESSION_BC'], ['FOR_BLOCK11', 'BREAK_CONTINUE'], ['FOR_BLOCK15', 'S'], ['FOR_BLOCK18', 'IF_BLOCK_BC'], ['FOR_BLOCK23', 'S'], ['FOR_BLOCK28', 'S'], ['FOR_BLOCK33', 'ELIF_BLOCK'], ['FOR_BLOCK38', 'IF_BLOCK'], ['FOR_BLOCK43', 'ELSE_BLOCK'], ['FOR_BLOCK46', 'IF_BLOCK_COND'], ['FOR_BLOCK49', 'IF_BLOCK_BC'], ['FOR_BLOCK54', 'IF_BLOCK_K'], ['FOR_BLOCK59', 'IF_BLOCK_BC'], ['FOR_BLOCK62', 'IF_BLOCK_BC']], 'METHOD_PARAM': [['VALUE'], ['ASSIGN']], 'METHOD': [['METHOD3', 'RIGHT_RB']], 'DEF_BLOCK': [['DEF_BLOCK3', 'ONE_LINE'], ['DEF_BLOCK7', 'S']], 'RETURN': [['T16', 'BOOLEAN'], ['T16', 'VARIABLE'], ['T16', 'DATA_TYPE']], 'PARAM_CLASS': [['PARAM_CLASS1', 'RIGHT_RB'], ['LEFT_RB', 'RIGHT_RB']], 'CLASS': [['CLASS2', 'ONE_LINE'], ['CLASS5', 'S']], 'CLASS_OBJECT': [['VARIABLE', 'PARAM_CLASS']], 'IMPORT': [['T18', 'MODULE'], ['IMPORT2', 'VARIABLE']], 'FROM': [['FROM1', 'IMPORT'], ['RAISE', 'RAISE'], ['RAISE', 'VARIABLE'], ['RAISE3', 'RIGHT_RB'], ['RAISE8', 'VARIABLE'], ['RAISE13', 'NONE']], 'T1': [['=']], 'T2': [['!']], 'T3': [['int']], 'T4': [['str']], 'T5': [['float']], 'T6': [['input']], 'T7': [['print']], 'T8': [['comment']], 'T9': [['if']], 'T10': [['else']], 'T11': [['elif']], 'T12': [['while']], 'T13': [['range']], 'T14': [['for']], 'T15': [['def']], 'T16': [['return']], 'T17': [['class']], 'T18': [['import']], 'T19': [['from']], 'T20': [['variable']], 'T21': [['>']], 'T22': [['<']], 'T23': [['is']], 'T24': [['not']], 'T25': [['*']], 'T26': [['/']], 'T27': [['-']], 'T28': [['integer']], 'T29': [['string']], 'T30': [['break']], 'T31': [['continue']], 'T32': [['+']], 'T33': [['%']], 'S1': [['S', 'NEWLINE']], 'VARIABLE1': [['T20', 'PERIOD']], 'MULTI_CLASS1': [['VALUE', 'COMMA']], 'MULTI_VAR1': [['VARIABLE', 'COMMA']], 'MULTI_TYPE1': [['TYPE', 'COMMA']], 'MULTI_TYPE2': [['LIST', 'COMMA']], 'WITH1': [['VARIABLE', 'PARAM']], 'WITH2': [['WITH1', 'AS']], 'WITH3': [['WITH2', 'VARIABLE']], 'WITH4': [['WITH3', 'COLON']], 'WITH5': [['WITH4', 'NEWLINE']], 'FLOAT1': [['T28', 'PERIOD']], 'FLOAT2': [['T27', 'T28']], 'FLOAT3': [['FLOAT2', 'PERIOD']], 'BOOLEAN1': [['LEFT_RB', 'BOOLEAN']], 'BOOLEAN2': [['LEFT_RB', 'BOOLEAN_OPERAND']], 'BOOLEAN3': [['T24', 'LEFT_RB']], 'BOOLEAN4': [['BOOLEAN3', 'BOOLEAN']], 'BOOLEAN5': [['BOOLEAN_OPERAND', 'RELATIONAL_OP']], 'BOOLEAN6': [['BOOLEAN', 'OR_AND']], 'BOOLEAN7': [['LEFT_RB', 'BOOLEAN']], 'BOOLEAN8': [['BOOLEAN', 'RIGHT_RB']], 'BOOLEAN9': [['BOOLEAN8', 'OR_AND']], 'BOOLEAN10': [['BOOLEAN9', 'LEFT_RB']], 'BOOLEAN11': [['BOOLEAN10', 'BOOLEAN']], 'EXPRESSION_ONE_LINE1': [['BOOLEAN', 'COLON']], 'EXPRESSION_ONE_LINE2': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_ONE_LINE3': [['EXPRESSION_ONE_LINE2', 'RIGHT_RB']], 'EXPRESSION_ONE_LINE4': [['EXPRESSION_ONE_LINE3', 'COLON']], 'EXPRESSION1': [['BOOLEAN', 'COLON']], 'EXPRESSION2': [['EXPRESSION1', 'NEWLINE']], 'EXPRESSION3': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION4': [['EXPRESSION3', 'RIGHT_RB']], 'EXPRESSION5': [['EXPRESSION4', 'COLON']], 'EXPRESSION6': [['EXPRESSION5', 'NEWLINE']], 'EXPRESSION7': [['BOOLEAN', 'COLON']], 'EXPRESSION8': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION9': [['EXPRESSION8', 'RIGHT_RB']], 'EXPRESSION10': [['EXPRESSION9', 'COLON']], 'EXPRESSION_COND_BC1': [['BOOLEAN', 'COLON']], 'EXPRESSION_COND_BC2': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_COND_BC3': [['EXPRESSION_COND_BC2', 'RIGHT_RB']], 'EXPRESSION_COND_BC4': [['EXPRESSION_COND_BC3', 'COLON']], 'PARAM1': [['LEFT_RB', 'MULTI_VAR']], 'CASTING1': [['T3', 'LEFT_RB']], 'CASTING2': [['CASTING1', 'T29']], 'CASTING3': [['T4', 'LEFT_RB']], 'CASTING4': [['CASTING3', 'INTEGER']], 'CASTING5': [['T3', 'LEFT_RB']], 'CASTING6': [['CASTING5', 'FLOAT']], 'CASTING7': [['T5', 'LEFT_RB']], 'CASTING8': [['CASTING7', 'INTEGER']], 'CASTING9': [['T4', 'LEFT_RB']], 'CASTING10': [['CASTING9', 'FLOAT']], 'CASTING11': [['T5', 'LEFT_RB']], 'CASTING12': [['CASTING11', 'T29']], 'INPUT1': [['T6', 'LEFT_RB']], 'INPUT2': [['INPUT1', 'T29']], 'PRINT1': [['T7', 'LEFT_RB']], 'PRINT2': [['PRINT1', 'VARIABLE']], 'PRINT3': [['T7', 'LEFT_RB']], 'PRINT4': [['PRINT3', 'TYPE']], 'PRINT5': [['T7', 'LEFT_RB']], 'PRINT6': [['PRINT5', 'BOOLEAN']], 'TUPLE1': [['LEFT_RB', 'MULTI_TYPE']], 'LIST1': [['LEFT_SB', 'MULTI_TYPE']], 'ARITH1': [['ARITH', 'ARITHMETIC_OP']], 'ARR_ELMT1': [['VARIABLE', 'LEFT_SB']], 'ARR_ELMT2': [['ARR_ELMT1', 'VARIABLE']], 'ARR_ELMT3': [['VARIABLE', 'LEFT_SB']], 'ARR_ELMT4': [['ARR_ELMT3', 'INTEGER']], 'ARR_ELMT5': [['VARIABLE', 'LEFT_SB']], 'ARR_ELMT6': [['ARR_ELMT5', 'ARITH']], 'BREAK_CONTINUE1': [['S', 'NEWLINE']], 'BREAK_CONTINUE2': [['S', 'NEWLINE']], 'BREAK_CONTINUE3': [['T30', 'NEWLINE']], 'BREAK_CONTINUE4': [['T31', 'NEWLINE']], 'BREAK_CONTINUES': [['NEWLINE', 'T30']], 'BREAK_CONTINUE6': [['BREAK_CONTINUES', 'NEWLINE']], 'BREAK_CONTINUE7': [['NEWLINE', 'T31']], 'BREAK_CONTINUE8': [['BREAK_CONTINUE7', 'NEWLINE']], 'ASSIGN1': [['ASSIGN_DEST', 'T1']], 'ASSIGN2': [['ASSIGN_DEST', 'T1']], 'ASSIGN3': [['ASSIGN_DEST', 'T1']], 'ASSIGN4': [['ASSIGN_DEST', 'T1']], 'ASSIGNS': [['ASSIGN_DEST', 'T1']], 'SELF_ARITH1': [['ASSIGN_DEST', 'T32']], 'SELF_ARITH2': [['SELF_ARITH1', 'T1']], 'SELF_ARITH3': [['ASSIGN_DEST', 'T32']], 'SELF_ARITH4': [['SELF_ARITH3', 'T1']], 'SELF_ARITH5': [['ASSIGN_DEST', 'T27']], 'SELF_ARITH6': [['SELF_ARITH5', 'T1']], 'SELF_ARITH7': [['ASSIGN_DEST', 'T25']], 'SELF_ARITH8': [['SELF_ARITH7', 'T1']], 'SELF_ARITH9': [['ASSIGN_DEST', 'T26']], 'SELF_ARITH10': [['SELF_ARITH9', 'T1']], 'SELF_ARITH11': [['ASSIGN_DEST', 'T33']], 'SELF_ARITH12': [['SELF_ARITH11', 'T1']], 'SELF_ARITH13': [['ASSIGN_DEST', 'T26']], 'SELF_ARITH14': [['SELF_ARITH13', 'T26']], 'SELF_ARITH15': [['SELF_ARITH14', 'T1']], 'SELF_ARITH16': [['ASSIGN_DEST', 'T25']], 'SELF_ARITH17': [['SELF_ARITH16', 'T25']], 'SELF_ARITH18': [['SELF_ARITH17', 'T1']], 'IF_ONE_LINE1': [['VALUE', 'T9']], 'IF_ONE_LINE2': [['IF_ONE_LINE1', 'BOOLEAN']], 'IF_ONE_LINE3': [['IF_ONE_LINE2', 'T10']], 'IF_ONE_LINE4': [['VARIABLE', 'T1']], 'IF_ONE_LINE5': [['IF_ONE_LINE4', 'VALUE']], 'IF_ONE_LINE6': [['IF_ONE_LINE5', 'T9']], 'IF_ONE_LINE7': [['IF_ONE_LINE6', 'BOOLEAN']], 'IF_ONE_LINE8': [['IF_ONE_LINE7', 'T10']], 'IF_BLOCK1': [['T9', 'EXPRESSION']], 'IF_BLOCK2': [['T9', 'EXPRESSION']], 'IF_BLOCK3': [['IF_BLOCK2', 'T10']], 'IF_BLOCK4': [['IF_BLOCK3', 'COLON']], 'IF_BLOCK5': [['IF_BLOCK4', 'NEWLINE']], 'IF_BLOCK6': [['T9', 'EXPRESSION']], 'IF_BLOCK7': [['IF_BLOCK6', 'T10']], 'IF_BLOCK8': [['IF_BLOCK7', 'COLON']], 'ELIF_BLOCK1': [['T11', 'EXPRESSION']], 'ELIF_BLOCK2': [['T11', 'EXPRESSION']], 'ELIF_BLOCK3': [['ELIF_BLOCK2', 'T10']], 'ELIF_BLOCK4': [['ELIF_BLOCK3', 'COLON']], 'ELIF_BLOCKS': [['ELIF_BLOCK4', 'NEWLINE']], 'ELIF_BLOCK6': [['T11', 'EXPRESSION']], 'ELIF_BLOCK7': [['ELIF_BLOCK6', 'T10']], 'ELIF_BLOCK8': [['ELIF_BLOCK7', 'COLON']], 'ELSE_BLOCK1': [['T10', 'COLON']], 'ELSE_BLOCK2': [['ELSE_BLOCK1', 'NEWLINE']], 'ELSE_BLOCK3': [['T10', 'COLON']], 'IF_BLOCK_COND1': [['T9', 'EXPRESSION']], 'IF_BLOCK_COND2': [['IF_BLOCK', 'NEWLINE']], 'IF_BLOCK_COND3': [['IF_BLOCK_COND', 'NEWLINE']], 'EXPRESSION_BC1': [['BOOLEAN', 'COLON']], 'EXPRESSION_BC2': [['EXPRESSION_BC1', 'NEWLINE']], 'EXPRESSION_BC3': [['EXPRESSION_BC2', 'BREAK_CONTINUE']], 'EXPRESSION_BC4': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_BC5': [['EXPRESSION_BC4', 'RIGHT_RB']], 'EXPRESSION_BC6': [['EXPRESSION_BC5', 'COLON']], 'EXPRESSION_BC7': [['EXPRESSION_BC6', 'NEWLINE']], 'EXPRESSION_BC8': [['EXPRESSION_BC7', 'BREAK_CONTINUE']], 'EXPRESSION_BC9': [['BOOLEAN', 'COLON']], 'EXPRESSION_BC10': [['EXPRESSION_BC9', 'BREAK_CONTINUE']], 'EXPRESSION_BC11': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_BC12': [['EXPRESSION_BC11', 'RIGHT_RB']], 'EXPRESSION_BC13': [['EXPRESSION_BC12', 'COLON']], 'EXPRESSION_BC14': [['EXPRESSION_BC13', 'BREAK_CONTINUE']], 'EXPRESSION_BC15': [['BOOLEAN',

```

```

'COLON']], 'EXPRESSION_BC16': [['EXPRESSION_BC15', 'NEWLINE']], 'EXPRESSION_BC17': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_BC18': [['EXPRESSION_BC17', 'RIGHT_RB']], 'EXPRESSION_BC19': [['EXPRESSION_BC18', 'COLON']], 'EXPRESSION_BC20': [['EXPRESSION_BC19', 'NEWLINE']], 'EXPRESSION_BC21': [['BOOLEAN', 'COLON']], 'EXPRESSION_BC22': [['LEFT_RB', 'BOOLEAN']], 'EXPRESSION_BC23': [['EXPRESSION_BC22', 'RIGHT_RB']], 'EXPRESSION_BC24': [['EXPRESSION_BC23', 'COLON']], 'IF_ONE_LINE_BC1': [['VALUE', 'T9']], 'IF_ONE_LINE_BC2': [['IF_ONE_LINE_BC1', 'BOOLEAN']], 'IF_ONE_LINE_BC3': [['IF_ONE_LINE_BC2', 'T10']], 'IF_ONE_LINE_BC4': [['VALUE', 'T9']], 'IF_ONE_LINE_BC5': [['IF_ONE_LINE_BC4', 'BOOLEAN']], 'IF_ONE_LINE_BC6': [['IF_ONE_LINE_BC5', 'T10']], 'IF_BLOCK_BC1': [['T9', 'EXPRESSION_COND_BC']], 'IF_BLOCK_BC2': [['IF_BLOCK', 'S']], 'IF_BLOCK_BC3': [['IF_BLOCK_BC2', 'BREAK_CONTINUE']], 'IF_BLOCK_BC4': [['T9', 'EXPRESSION']], 'IF_BLOCK_BC5': [['T9', 'EXPRESSION']], 'IF_BLOCK_BC6': [['IF_BLOCK_BC5', 'NEWLINE']], 'IF_BLOCK_BC7': [['IF_BLOCK', 'NEWLINE']], 'ELIF_BLOCK_BC1': [['T11', 'EXPRESSION']], 'ELSE_BLOCK_BC1': [['T10', 'COLON']], 'ELSE_BLOCK_BC2': [['ELSE_BLOCK_BC1', 'NEWLINE']], 'ELSE_BLOCK_BC3': [['T10', 'COLON']], 'ELSE_BLOCK_BC4': [['T10', 'COLON']], 'ELSE_BLOCK_BC5': [['ELSE_BLOCK_BC4', 'NEWLINE']], 'ELSE_BLOCK_BC6': [['ELSE_BLOCK_BC5', 'S']], 'WHILE_BLOCK1': [['T12', 'EXPRESSION']], 'WHILE_BLOCK2': [['WHILE_BLOCK1', 'IF_BC']], 'WHILE_BLOCK3': [['T12', 'EXPRESSION']], 'WHILE_BLOCK4': [['T12', 'EXPRESSION_BC']], 'WHILE_BLOCKS': [['WHILE_BLOCK4', 'NEWLINE']], 'WHILE_BLOCK6': [['T12', 'EXPRESSION']], 'WHILE_BLOCK7': [['T12', 'EXPRESSION']], 'WHILE_BLOCK8': [['WHILE_BLOCK7', 'IF_BLOCK_BC']], 'WHILE_BLOCK9': [['WHILE_BLOCK8', 'NEWLINE']], 'WHILE_BLOCK10': [['T12', 'EXPRESSION']], 'WHILE_BLOCK11': [['WHILE_BLOCK10', 'IF_BLOCK_BC']], 'WHILE_BLOCK12': [['WHILE_BLOCK11', 'NEWLINE']], 'WHILE_BLOCK13': [['T12', 'EXPRESSION']], 'WHILE_BLOCK14': [['WHILE_BLOCK13', 'IF_ONE_LINE_BC']], 'WHILE_BLOCK15': [['WHILE_BLOCK14', 'NEWLINE']], 'WHILE_BLOCK16': [['T12', 'EXPRESSION']], 'WHILE_BLOCK17': [['WHILE_BLOCK16', 'IF_BLOCK_BC']], 'WHILE_BLOCK18': [['WHILE_BLOCK17', 'NEWLINE']], 'WHILE_BLOCK19': [['T12', 'EXPRESSION']], 'WHILE_BLOCK20': [['WHILE_BLOCK19', 'NEWLINE']], 'WHILE_BLOCK21': [['T12', 'EXPRESSION_COND_BC']], 'WHILE_BLOCK22': [['T12', 'EXPRESSION_COND_BC']], 'WHILE_BLOCK23': [['WHILE_BLOCK22', 'IF_BLOCK_BC']], 'WHILE_BLOCK24': [['WHILE_BLOCK23', 'NEWLINE']], 'WHILE_BLOCK25': [['T12', 'EXPRESSION_COND_BC']], 'WHILE_BLOCK26': [['WHILE_BLOCK25', 'IF_BLOCK_BC']], 'WHILE_BLOCK27': [['WHILE_BLOCK26', 'NEWLINE']], 'WHILE_BLOCK28': [['T12', 'EXPRESSION_COND_BC']], 'WHILE_BLOCK29': [['WHILE_BLOCK28', 'IF_BLOCK']], 'WHILE_BLOCK30': [['WHILE_BLOCK29', 'NEWLINE']], 'WHILE_BLOCK31': [['T12', 'EXPRESSION']], 'RANGE_PARAM1': [['T13', 'LEFT_RB']], 'RANGE_PARAM2': [['RANGE_PARAM1', 'RANGE_TYPE']], 'RANGE_PARAM3': [['T13', 'LEFT_RB']], 'RANGE_PARAM4': [['RANGE_PARAM3', 'RANGE_TYPE']], 'RANGE_PARAMS': [['RANGE_PARAM4', 'COMMA']], 'RANGE_PARAM6': [['RANGE_PARAMS', 'RANGE_TYPE']], 'RANGE_PARAM7': [['T13', 'LEFT_RB']], 'RANGE_PARAM8': [['RANGE_PARAM7', 'RANGE_TYPE']], 'RANGE_PARAM9': [['RANGE_PARAM8', 'COMMA']], 'RANGE_PARAM10': [['RANGE_PARAM9', 'RANGE_TYPE']], 'RANGE_PARAM11': [['RANGE_PARAM10', 'COMMA']], 'RANGE_PARAM12': [['RANGE_PARAM11', 'RANGE_TYPE']], 'FOR_PARAM1': [['VARIABLE', 'IN']], 'FOR_PARAM2': [['VARIABLE', 'IN']], 'FOR_BLOCK1': [['T14', 'FOR_PARAM']], 'FOR_BLOCK2': [['FOR_BLOCK1', 'COLON']], 'FOR_BLOCK3': [['T14', 'FOR_PARAM']], 'FOR_BLOCK4': [['FOR_BLOCK3', 'COLON']], 'FOR_BLOCK5': [['FOR_BLOCK4', 'NEWLINE']], 'FOR_BLOCK6': [['T14', 'FOR_PARAM']], 'FOR_BLOCK7': [['FOR_BLOCK6', 'COLON']], 'FOR_BLOCK8': [['FOR_BLOCK7', 'NEWLINE']], 'FOR_BLOCK9': [['T14', 'FOR_PARAM']], 'FOR_BLOCK10': [['FOR_BLOCK9', 'COLON']], 'FOR_BLOCK11': [['FOR_BLOCK10', 'NEWLINE']], 'FOR_BLOCK12': [['T14', 'FOR_PARAM']], 'FOR_BLOCK13': [['FOR_BLOCK12', 'COLON']], 'FOR_BLOCK14': [['FOR_BLOCK13', 'NEWLINE']], 'FOR_BLOCK15': [['FOR_BLOCK14', 'NEWLINE']], 'FOR_BLOCK16': [['T14', 'FOR_PARAM']], 'FOR_BLOCK17': [['FOR_BLOCK16', 'COLON']], 'FOR_BLOCK18': [['FOR_BLOCK17', 'NEWLINE']], 'FOR_BLOCK19': [['T14', 'FOR_PARAM']], 'FOR_BLOCK20': [['FOR_BLOCK19', 'COLON']], 'FOR_BLOCK21': [['FOR_BLOCK20', 'NEWLINE']], 'FOR_BLOCK22': [['FOR_BLOCK21', 'IF_BLOCK_BC']], 'FOR_BLOCK23': [['FOR_BLOCK22', 'NEWLINE']], 'FOR_BLOCK24': [['T14', 'FOR_PARAM']], 'FOR_BLOCK25': [['FOR_BLOCK24', 'COLON']], 'FOR_BLOCK26': [['FOR_BLOCK25', 'NEWLINE']], 'FOR_BLOCK27': [['FOR_BLOCK26', 'IF_ONE_LINE_BC']], 'FOR_BLOCK28': [['FOR_BLOCK27', 'NEWLINE']], 'FOR_BLOCK29': [['T14', 'FOR_PARAM']], 'FOR_BLOCK30': [['FOR_BLOCK29', 'COLON']], 'FOR_BLOCK31': [['FOR_BLOCK30', 'NEWLINE']], 'FOR_BLOCK32': [['FOR_BLOCK31', 'IF_BLOCK_BC']], 'FOR_BLOCK33': [['FOR_BLOCK32', 'NEWLINE']], 'FOR_BLOCK34': [['T14', 'FOR_PARAM']], 'FOR_BLOCK35': [['FOR_BLOCK34', 'COLON']], 'FOR_BLOCK36': [['FOR_BLOCK35', 'NEWLINE']], 'FOR_BLOCK37': [['FOR_BLOCK36', 'IF_BLOCK_BC']], 'FOR_BLOCK38': [['FOR_BLOCK37', 'NEWLINE']], 'FOR_BLOCK39': [['T14', 'FOR_PARAM']], 'FOR_BLOCK40': [['FOR_BLOCK39', 'COLON']], 'FOR_BLOCK41': [['FOR_BLOCK40', 'NEWLINE']], 'FOR_BLOCK42': [['FOR_BLOCK41', 'IF_BLOCK_BC']], 'FOR_BLOCK43': [['FOR_BLOCK42', 'NEWLINE']], 'FOR_BLOCK44': [['T14', 'FOR_PARAM']], 'FOR_BLOCK45': [['FOR_BLOCK44', 'COLON']], 'FOR_BLOCK46': [['FOR_BLOCK45', 'NEWLINE']], 'FOR_BLOCK47': [['T14', 'FOR_PARAM']], 'FOR_BLOCK48': [['FOR_BLOCK47', 'COLON']], 'FOR_BLOCK49': [['T14', 'FOR_PARAM']], 'FOR_BLOCK50': [['T14', 'FOR_PARAM']], 'FOR_BLOCK51': [['FOR_BLOCK50', 'COLON']], 'FOR_BLOCK52': [['FOR_BLOCK51', 'NEWLINE']], 'FOR_BLOCK53': [['FOR_BLOCK52', 'IF_BLOCK_BC']], 'FOR_BLOCK54': [['FOR_BLOCK53', 'NEWLINE']], 'FOR_BLOCK55': [['T14', 'FOR_PARAM']], 'FOR_BLOCK56': [['FOR_BLOCK55', 'COLON']], 'FOR_BLOCK57': [['FOR_BLOCK56', 'NEWLINE']], 'FOR_BLOCK58': [['FOR_BLOCK57', 'IF_BLOCK']], 'FOR_BLOCK59': [['FOR_BLOCK58', 'NEWLINE']], 'FOR_BLOCK60': [['T14', 'FOR_PARAM']], 'FOR_BLOCK61': [['FOR_BLOCK60', 'COLON']], 'FOR_BLOCK62': [['FOR_BLOCK61', 'NEWLINE']], 'METHOD1': [['VARIABLE', 'PERIOD']], 'METHOD2': [['METHOD1', 'LEFT_RB']], 'METHOD3': [['METHOD2', 'METHOD_PARAM']], 'DEF_BLOCK1': [['T15', 'VARIABLE']], 'DEF_BLOCK2': [['DEF_BLOCK1', 'PARAM']], 'DEF_BLOCK3': [['DEF_BLOCK2', 'COLON']], 'DEF_BLOCK4': [['T15', 'VARIABLE']], 'DEF_BLOCKS': [['DEF_BLOCK4', 'PARAM']], 'DEF_BLOCK6': [['DEF_BLOCKS', 'COLON']], 'DEF_BLOCK7': [['DEF_BLOCK6', 'NEWLINE']], 'PARAM_CLASS1': [['LEFT_RB', 'MULTI_CLASS']], 'CLASS1': [['T17', 'VARIABLE']], 'CLASS2': [['CLASS1', 'COLON']], 'CLASS3': [['T17', 'VARIABLE']], 'CLASS4': [['CLASS3', 'COLON']], 'CLASS5': [['CLASS4', 'NEWLINE']], 'IMPORT1': [['IMPORT', 'MODULE']], 'IMPORT2': [['IMPORT1', 'AS']], 'FROM1': [['T19', 'MODULE']], 'RAISE1': [['RAISE', 'VARIABLE']], 'RAISE2': [['RAISE1', 'LEFT_RB']], 'RAISE3': [['RAISE2', 'T29']], 'RAISE4': [['RAISE', 'VARIABLE']], 'RAISE5': [['RAISE4', 'LEFT_RB']], 'RAISE6': [['RAISE5', 'T29']], 'RAISE7': [['RAISE6', 'RIGHT_RB']], 'RAISE8': [['RAISE7', 'T19']], 'RAISE9': [['RAISE', 'VARIABLE']], 'RAISE10': [['RAISE9', 'LEFT_RB']], 'RAISE11': [['RAISE10', 'T29']], 'RAISE12': [['RAISE11', 'RIGHT_RB']], 'RAISE13': [['RAISE12', 'T19']]

```

```

OCK33': [['FOR_BLOCK32', 'NEWLINE']], 'FOR_BLOCK34': [['T14', 'FOR_PARAM']], 'FOR_BLOCK35': [['FOR_BLOCK34', 'COLON']], 'FOR_BLOCK36': [['FOR_BLOCK35', 'NEWLINE']], 'FOR_BLOCK37': [['FOR_BLOCK36', 'IF_BLOCK_BC']], 'FOR_BLOCK38': [['FOR_BLOCK37', 'NEWLINE']], 'FOR_BLOCK39': [['T14', 'FOR_PARAM']], 'FOR_BLOCK40': [['FOR_BLOCK39', 'COLON']], 'FOR_BLOCK41': [['FOR_BLOCK40', 'NEWLINE']], 'FOR_BLOCK42': [['FOR_BLOCK41', 'IF_BLOCK_BC']], 'FOR_BLOCK43': [['FOR_BLOCK42', 'NEWLINE']], 'FOR_BLOCK44': [['T14', 'FOR_PARAM']], 'FOR_BLOCK45': [['FOR_BLOCK44', 'COLON']], 'FOR_BLOCK46': [['FOR_BLOCK45', 'NEWLINE']], 'FOR_BLOCK47': [['T14', 'FOR_PARAM']], 'FOR_BLOCK48': [['FOR_BLOCK47', 'COLON']], 'FOR_BLOCK49': [['T14', 'FOR_PARAM']], 'FOR_BLOCK50': [['T14', 'FOR_PARAM']], 'FOR_BLOCK51': [['FOR_BLOCK50', 'COLON']], 'FOR_BLOCK52': [['FOR_BLOCK51', 'NEWLINE']], 'FOR_BLOCK53': [['FOR_BLOCK52', 'IF_BLOCK_BC']], 'FOR_BLOCK54': [['FOR_BLOCK53', 'NEWLINE']], 'FOR_BLOCK55': [['T14', 'FOR_PARAM']], 'FOR_BLOCK56': [['FOR_BLOCK55', 'COLON']], 'FOR_BLOCK57': [['FOR_BLOCK56', 'NEWLINE']], 'FOR_BLOCK58': [['FOR_BLOCK57', 'IF_BLOCK']], 'FOR_BLOCK59': [['FOR_BLOCK58', 'NEWLINE']], 'FOR_BLOCK60': [['T14', 'FOR_PARAM']], 'FOR_BLOCK61': [['FOR_BLOCK60', 'COLON']], 'FOR_BLOCK62': [['FOR_BLOCK61', 'NEWLINE']], 'METHOD1': [['VARIABLE', 'PERIOD']], 'METHOD2': [['METHOD1', 'LEFT_RB']], 'METHOD3': [['METHOD2', 'METHOD_PARAM']], 'DEF_BLOCK1': [['T15', 'VARIABLE']], 'DEF_BLOCK2': [['DEF_BLOCK1', 'PARAM']], 'DEF_BLOCK3': [['DEF_BLOCK2', 'COLON']], 'DEF_BLOCK4': [['T15', 'VARIABLE']], 'DEF_BLOCKS': [['DEF_BLOCK4', 'PARAM']], 'DEF_BLOCK6': [['DEF_BLOCKS', 'COLON']], 'DEF_BLOCK7': [['DEF_BLOCK6', 'NEWLINE']], 'PARAM_CLASS1': [['LEFT_RB', 'MULTI_CLASS']], 'CLASS1': [['T17', 'VARIABLE']], 'CLASS2': [['CLASS1', 'COLON']], 'CLASS3': [['T17', 'VARIABLE']], 'CLASS4': [['CLASS3', 'COLON']], 'CLASS5': [['CLASS4', 'NEWLINE']], 'IMPORT1': [['IMPORT', 'MODULE']], 'IMPORT2': [['IMPORT1', 'AS']], 'FROM1': [['T19', 'MODULE']], 'RAISE1': [['RAISE', 'VARIABLE']], 'RAISE2': [['RAISE1', 'LEFT_RB']], 'RAISE3': [['RAISE2', 'T29']], 'RAISE4': [['RAISE', 'VARIABLE']], 'RAISE5': [['RAISE4', 'LEFT_RB']], 'RAISE6': [['RAISE5', 'T29']], 'RAISE7': [['RAISE6', 'RIGHT_RB']], 'RAISE8': [['RAISE7', 'T19']], 'RAISE9': [['RAISE', 'VARIABLE']], 'RAISE10': [['RAISE9', 'LEFT_RB']], 'RAISE11': [['RAISE10', 'T29']], 'RAISE12': [['RAISE11', 'RIGHT_RB']], 'RAISE13': [['RAISE12', 'T19']]

```

BAB III

Implementasi dan Pengujian

3.1 Spesifikasi Teknis Program

3.1.1 File CFGtoCNF.py

File CFGtoCNF.py berisi prosedur dan fungsi untuk mengkonversi CFG menjadi CNF.

Tabel 3.1 Spesifikasi File Program CFGtoCNF.py

| No | Fungsi/Prosedur | Tujuan |
|----|----------------------|---|
| 1 | readCFGFile | Membaca CFG dari file txt |
| 2 | isVar | Mengecek apakah suatu simbol merupakan variabel |
| 3 | removeUnitProduction | Menghilangkan produksi unit pada CFG |
| 4 | get_key | Mendapatkan variabel dari suatu simbol pada aturan produksi |
| 5 | CFGtoCNF | Mengkonversi CFG menjadi CNF |

3.1.2 File cyk.py

File cyk.py berisi prosedur dan fungsi untuk mengecek apakah suatu input yang sudah dijadikan ke dalam token-token termasuk ke dalam bahasa CFG.

Tabel 3.2 Spesifikasi File Program cyk.py

| No | Fungsi/Prosedur | Tujuan |
|----|-------------------|--|
| 1 | getRuleProduction | Mendapatkan variabel dari suatu simbol pada aturan produksi CFG yang sudah dikonversi ke CNF |
| 2 | concatRule | Mengembalikan hasil dari perkalian (konkat) himpunan rule satu dengan himpunan rule dua |
| 3 | cykAlgorithm | Mengembalikan himpunan set hasil dari pemeriksaan input token dengan algoritma cyk |

3.1.3 File dfa.py

File dfa.py berisi fungsi untuk memeriksa kevalidan nama variabel menggunakan DFA pada bagian 2.3.

Tabel 3.3 Spesifikasi File Program dfa.py

| No | Fungsi/Prosedur | Tujuan |
|----|-----------------|---|
| 1 | isVarNameValid | Memeriksa kevalidan nama variabel dengan implementasi DFA. Mengembalikan True jika nama variabel valid. |

3.1.4 File tokens.py

File tokens.py berisi fungsi untuk mengubah kode program yang ingin dicek ke dalam token. Token yang digunakan sama dengan simbol terminal pada CFG.

Tabel 3.4 Spesifikasi File Program tokens.py

| No | Fungsi/Prosedur | Tujuan |
|----|-----------------|--|
| 2 | readPythonFile | Mengubah kode program yang ingin dicek ke dalam token. |

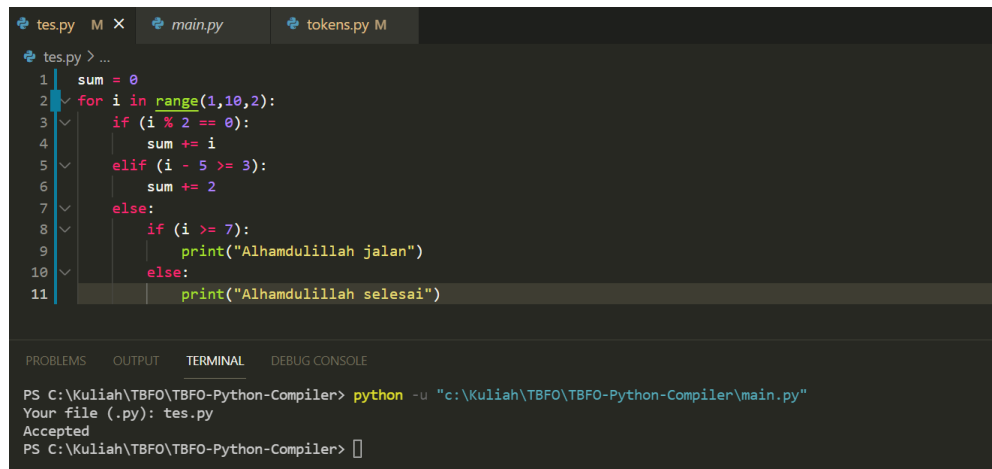
3.1.5 File main.py

File main.py berisi intruksi-intruksi dari program utama. Pada saat menjalankan file main.py, pengguna diminta untuk memasukkan nama file (.py) yang akan diperiksa sintaksnya. Jika sintaks benar, program akan menuliskan keterangan "Accepted", jika tidak program akan menuliskan keterangan "Syntax Error".

3.2 Pengujian Program

Pada bagian ini, kami melakukan uji coba untuk berbagai contoh kasus.

3.2.1 Kasus perulangan bersarang



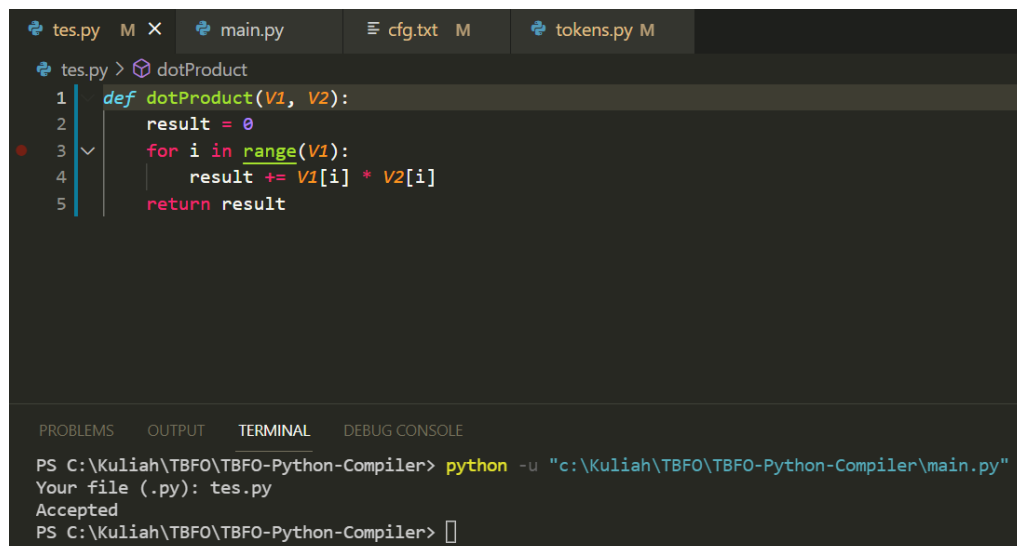
```
tes.py M X main.py tokens.py M
tes.py > ...
1 sum = 0
2 for i in range(1,10,2):
3     if (i % 2 == 0):
4         sum += i
5     elif (i - 5 >= 3):
6         sum += 2
7     else:
8         if (i >= 7):
9             print("Alhamdulillah jalan")
10        else:
11            print("Alhamdulillah selesai")

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Kuliah\TBFO\TBFO-Python-Compiler> python -u "c:\Kuliah\TBFO\TBFO-Python-Compiler\main.py"
Your file (.py): tes.py
Accepted
PS C:\Kuliah\TBFO\TBFO-Python-Compiler> 
```

Gambar 3.1 Hasil pengujian kasus perulangan bersarang

Pada contoh kasus ini, kami melakukan pengujian terhadap contoh kasus perulangan dan percabangan sekaligus. Di dalam *for loop* terdapat percabangan bersarang. Terlihat bahwa sintaks yang digunakan sudah benar sehingga keluaran yang dihasilkan adalah "Accepted".

3.2.2 Kasus fungsi dengan operasi array satu dimensi



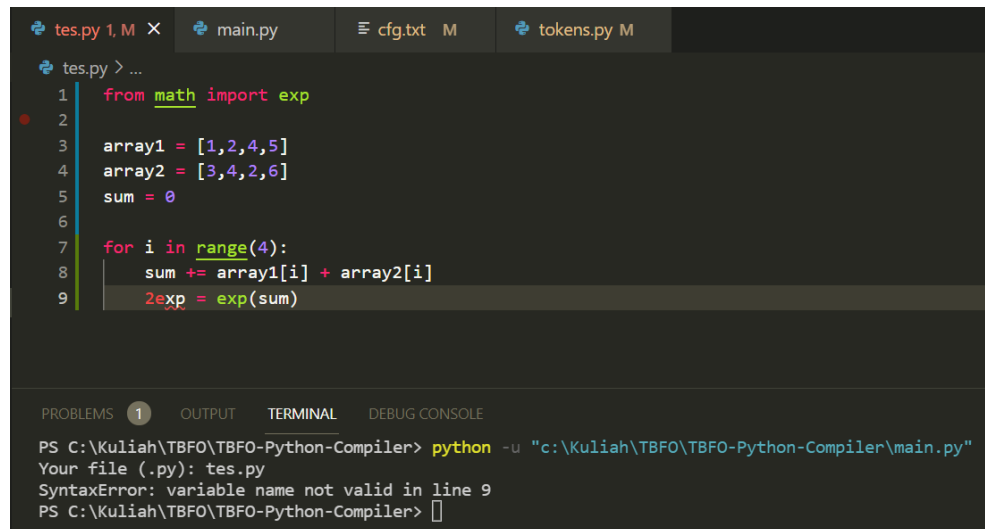
```
tes.py M X main.py cfg.txt M tokens.py M
tes.py > dotProduct
1 def dotProduct(V1, V2):
2     result = 0
3     for i in range(V1):
4         result += V1[i] * V2[i]
5     return result

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS C:\Kuliah\TBFO\TBFO-Python-Compiler> python -u "c:\Kuliah\TBFO\TBFO-Python-Compiler\main.py"
Your file (.py): tes.py
Accepted
PS C:\Kuliah\TBFO\TBFO-Python-Compiler> 
```

Gambar 3.2 Hasil pengujian kasus fungsi dengan operasi array satu dimensi

Pada contoh kasus ini, kami melakukan pengujian terhadap contoh kasus fungsi dengan operasi *array* satu dimensi. Di dalam fungsi operasi perkalian dot seperti pada vektor. Terlihat bahwa sintaks yang digunakan sudah benar sehingga keluaran yang dihasilkan adalah “Accepted”.

3.2.3 Kasus kesalahan nama variabel



```
tes.py 1, M X main.py cfg.txt M tokens.py M
tes.py > ...
1 from math import exp
2
3 array1 = [1,2,4,5]
4 array2 = [3,4,2,6]
5 sum = 0
6
7 for i in range(4):
8     sum += array1[i] + array2[i]
9     2exp = exp(sum)
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Kuliah\TBFO\TBFO-Python-Compiler> python -u "c:\Kuliah\TBFO\TBFO-Python-Compiler\main.py"

Your file (.py): tes.py

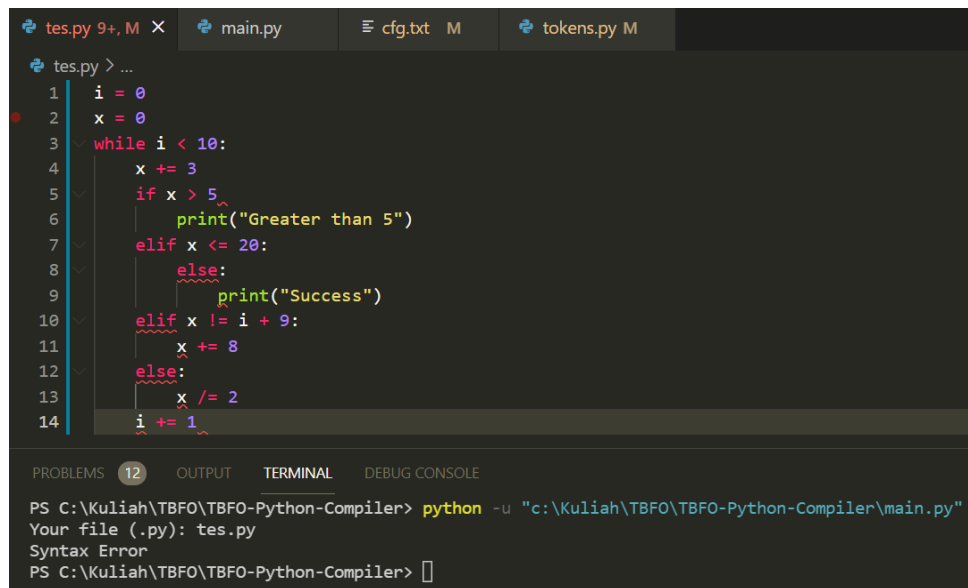
SyntaxError: variable name not valid in line 9

PS C:\Kuliah\TBFO\TBFO-Python-Compiler>

Gambar 3.3 Hasil pengujian kasus kesalahan nama variabel

Pada contoh kasus ini, kami melakukan pengujian terhadap contoh kasus kesalahan nama variabel. Terlihat pada baris ke-9 terdapat kesalahan nama variabel, yaitu “2exp”. Seharusnya nama variabel tidak diawali dengan angka. Terlihat bahwa keluaran yang dihasilkan adalah keterangan bahwa terjadi kesalahan nama variabel pada baris ke-9.

3.2.4 Kasus kesalahan sintaks percabangan



```
tes.py 9+, M X main.py cfg.txt M tokens.py M
tes.py > ...
1 i = 0
2 x = 0
3 while i < 10:
4     x += 3
5     if x > 5:
6         print("Greater than 5")
7     elif x <= 20:
8         else:
9             print("Success")
10    elif x != i + 9:
11        x += 8
12    else:
13        x /= 2
14    i += 1
```

PROBLEMS 12 OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Kuliah\TBFO\TBFO-Python-Compiler> python -u "c:\Kuliah\TBFO\TBFO-Python-Compiler\main.py"

Your file (.py): tes.py

Syntax Error

PS C:\Kuliah\TBFO\TBFO-Python-Compiler>

Gambar 3.4 Hasil pengujian kasus kesalahan sintaks percabangan

Pada contoh kasus ini, kami melakukan pengujian terhadap contoh kasus kesalahan sintaks percabangan. Terlihat pada baris ke-8 terdapat kesalahan sintaks. Kata kunci “else” harus didahului dengan “if”. Terlihat bahwa keluaran yang dihasilkan adalah keterangan bahwa terjadi kesalahan sintaks.

LINK GITHUB

<https://github.com/daffarg/TBFO-Python-Compiler>

REFERENSI

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley.

Chomsky Normal Form. Tutorialspoint. (n.d.). Retrieved November 23, 2021, from https://www.tutorialspoint.com/automata_theory/chomsky_normal_form.htm.

Automata Chomsky's normal form (CNF) - javatpoint. JavaT Point. (n.d.). Retrieved November 23, 2021, from <https://www.javatpoint.com/automata-chomskys-normal-form>

Algoritma Cyk - Unsyiah. Informatika Unsyiah. (n.d.). Retrieved November 23, 2021, from <http://informatika.unsyiah.ac.id/~viska/tba/algoritma.cyk.pdf>.

The CYK algorithm - computer science. CS UC Davis. (n.d.). Retrieved November 23, 2021, from <https://web.cs.ucdavis.edu/~rogaway/classes/120/winter12/CYK.pdf>.

What is python? executive summary. Python.org. (n.d.). Retrieved November 23, 2021, from <https://www.python.org/doc/essays/blurb/>.

PEMBAGIAN TUGAS

| | |
|--|--|
| 13520118 - Mohamad Daffa Argakoesoemah | <ul style="list-style-type: none">• CFG• Token• DFA• Main |
| 13520125 - Ikmal Alfaozi | <ul style="list-style-type: none">• CFG to CNF• CYK• Main |