Tugas Besar IF2124 Teori Bahasa Formal dan Otomata

Compiler Bahasa Python

Kelompok 19 Kelas K03 :

Muhammad Gilang R.	13520137
Willy Wilsen	13520160
Ghazian Tsabit Alkamil	13520165

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

TAHUN AJARAN 2021/2022

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Python adalah bahasa *interpreter* tingkat tinggi (*high-level*), dan juga *general-purpose*. Python diciptakan oleh Guido van Rossum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan *code readability* dengan penggunaan *whitespace*-nya. Python adalah bahasa multiparadigma karena mengimplementasi paradigma fungsional, imperatif, berorientasi objek, dan reflektif.

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh programmer. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa berjenis *interpreter* maupun *compiler*, keduanya pasti melakukan pemeriksaan sintaks. Perbedaannya terletak pada apa yang dilakukan setelah proses pemeriksaan (kompilasi/*compile*) tersebut selesai dilakukan.

Dibutuhkan *grammar* bahasa dan algoritma *parser* untuk melakukan kompilasi. Sudah sangat banyak *grammar* dan algoritma yang dikembangkan untuk menghasilkan *compiler* dengan performa yang tinggi. Terdapat CFG, CNF^{-e}, CNF^{+e}, 2NF, 2LF, dll untuk *grammar* yang dapat digunakan, dan terdapat LL(0), LL(1), CYK, Earley's Algorithm, LALR, GLR, Shift-reduce, SLR, LR(1), dll untuk algoritma yang dapat digunakan untuk melakukan *parsing*.

Pada tugas besar ini, implementasikanlah *compiler* untuk Python untuk *statement-statement* dan sintaks-sintaks bawaan Python. Gunakanlah konsep CFG untuk pengerjaan compiler yang mengevaluasi syntax program. Untuk nama variabel dalam program, gunakanlah FA.

BAB II

TEORI DASAR

1.1. Finite Automaa

Finite automata adalah mesin abstrak yang terdiri dari sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali Bahasa paling sederhana (Bahasa regular) dan dapat diimplementasikan secara nyata dengan sistem bisa berada di salah satu dari sejumlah berhingga konfirgurasi internal yang disebut state. Finite automata memiliki jumlah state yang banyaknya berhingga dan dapat berpindah-pindah dari satu state ke state yang lainnya. Menurut jenisnya, finite state automata dibagi 2, yaitu Deterministic Finite Automata (DFA) dan Non Deterministic Finite Automata (NFA).

Karakteristik Finite Automata

1. Deterministic Finite Automata (DFA)

   Menurut definisinya, Deterministic Finite Automata (DFA) merupakan finite automata yang pada satu input string hanya bisa ditransmisikan pada satu state lain. Pada DFA, terdapat 5 tuples pada komponennya, yaitu $\{Q, \Sigma, q, F, \delta\}$.

   Keterangan :

   Q : Himpunan semua states.

   Σ : Himpunan simbol input.

   q : Initial state.

   F : Himpunan final state.

   δ : Fungsi transisi, didefinisikan sebagai $\delta : Q \times \Sigma \rightarrow Q$.

2. Non Deterministic Finite Automata (NFA)

   Menurut definisinya, Non Deterministic Finite Automata (NFA) mirip dengan DFA, tetapi pada NFA, inputnya bisa bertransisi ke lebih dari satu state. Selain itu, pada NFA terdapat ε yang didefinisikan sebagai string kosong. Pada NFA, terdapat 5 tuples pada komponennya, yaitu $\{Q, \Sigma, q, F, \delta\}$.

   Keterangan :

   Q : Himpunan semua states.

   Σ : Himpunan simbol input.

   q : Initial state.

   F : Himpunan final state.

   δ : Fungsi transisi, didefinisikan sebagai $\delta : Q \times (\Sigma \cup \varepsilon) \rightarrow 2^Q$.

1.2. Context Free Grammar

Context Free Grammar adalah sebuah bahasa formal yang digunakan untuk menggeralisasikan semua kemungkinan string pada formal *language* yang diberikan. Secara umum Context Free Grammar didefinisikan sebagai 4 tuples, yaitu sebagai berikut.

G = (V, T, P, S)

Keterangan :

G : Didefinisikan sebagai *grammar*.

T : Definisikan sebagai sebuah himpunan terbatas dari *terminal symbol*.

V : Definisikan sebagai sebuah himpunan *rules production*.

S : Sebagai *start symbol*.

Pada pemakaiannya, CFG perlu disederhanakan dengan tujuan untuk melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tak perlu atau aturan produksi tak berarti. Berikut merupakan langkah-langkah dalam melakukan penyederhanaan CFG.

1. Eliminasi ε-production
2. Eliminasi unit production
3. Eliminasi useless symbol

Selain itu, pada CFG, start symbol dipakai untuk menurunkan string. Sehingga dapat diturunkan string secara berulang kali dengan mengganti non-terminal simbol di sisi kanan produksi, sampai semua non-terminal diganti dengan simbol terminal.

1.3. Algoritma Cocke-Younger-Kasami

Algoritma yang diciptakan oleh J. Cocke, DH. Younger, dan T. Kasami ini, menurut definisinya, Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma yang dipakai untuk mengecek apakah suatu *string* dapat diterima oleh suatu Bahasa Bebas Konteks (CFG). Adapun pada tugas besar kali ini, penulis mengonversikan CFG ke CNF terlebih dahulu sebelum masuk ke algoritma CYK. Pada pengkonversian yang disebut normalisasi ini, dilewati proses parsing, yaitu proses pembacaan untai dalam bahasa yang sesuai CFG tertentu, dimana proses ini harus mengikuti aturan produksi dalam CFG tersebut.

Dikarenakan CYK termasuk dalam bidang program dinamis, hal tersebut disebabkan algoritma ini dapat membangun tabel status dua dimensi pada saat melakukan *parsing*, yang dimana pada saat penentuan *parsing* selanjutnya dapat diturunkan atau dihasilkan dari parsing sebelumnya, hingga akhir *string*, sehingga independensi yang diinginkan dapat dicapai ketika *parsing*, dimana string yang telah valid atau tidak dapat dilihat dari array atau tabel terbentuk pada proses ini. Selain untuk mengetahui validitas *string* dalam suatu CFG, algoritma CYK yang telah dimodifikasi dapat dipergunakan untuk membangun *parse tree*.

Dalam pengaplikasian CYK, terdapat syarat-syarat yang dibentuk dari proses tersebut, yaitu sebagai berikut.

- Input : String dengan n simbol.
- Output : Valid/ tak valid.
- Struktur data : tabel n × n.
- Baris dengan indeks 0 sampai n – 1 (atau 1 – n dengan modifikasi).
- Kolom dengan indeks 1 sampai n.
- Sel [i, j] simbol yang termasuk dalam *string* input.

1.4. Syntax Python

Python yang dibuat oleh programmer Belanda Bernama Guido Van Rossum ini adalah salah satu bahasa pemrograman yang berparadigma prosedural. Secara teknis, python dapat melakukan eksekusi sejumlah instruksi multi guna secara langsung (interpretatif) dengan metode orientasi objek. Python juga merupakan bahasa pemrogramman tingkat tinggi, sehingga mudah dipahami oleh pemula yang baru mengenal pemrogramman.

Adapun *identifier* python adalah nama yang digunakan untuk mengidentifikasi variable, fungsi, kelas, modul, ataupun objek lainnya. Identifier diawali oleh sebuah huruf dari A sampai Z atau dari a sampai z atau garis bawah (_) diikuti oleh nol atau lebih huruf, garis bawah, dan angka (0 sampai 9). Pada python tidak diperbolehkan menggunakan karakter dengan tanda baca seperti @, $, dan % dalam proses pengidentifikasi. Selain itu, python adalah bahasa pemrogramman yang case sensitive terhadap huruf besar/kecil. Sehingga huruf kecil/besar harus benar-benar diperhatikan dalam bahasa pemrogramman ini.

Berikut adalah aturan penamaan pada Identifier Python.

1. Nama kelas dimulai dengan huruf besar. Semua pengenal lainnya dimulai dengan huruf kecil.
2. Identifier dimulai dengan satu garis bawah menunjukkan identifier bersifat pribadi.
3. Identifier dimulai dengan dua garis bawah di depan menunjukkan identifier bersifat sangat *private*.
4. Jika identifier diakhiri dengan dua garis bawah tambahan, menunjukkan identifier adalah sebuah bahasa untuk nama khusus.

BAB III

IMPLEMENTASI DAN PENGUJIAN

3.1. Impelementasi

Seluruh hasil implementasi program dapat dilihat melalui github :

https://github.com/TubesForLyfe/Tubes-TBFO

3.1.1. Hasil CFG dan FA

CFG yang telah dibuat terimplementasi pada pembuatan grammar sintax-sintax di python dan FA yang telah dibuat terimplementasi pada aturan penamaan variabelnya.

3.1.1.1. CFG

1. Main Content

START -> CONTENT

CONTENT -> CONTENT NEWLINE CONTENT | NEWLINE CONTENT | CONTENT NEWLINE | IF_CONTENT | ITERATE_CONTENT | CLASS_CONTENT | DEF_FUNC_CONTENT | ASS_OPERATION | FLOW_CONTENT | IMP_OPERATION | RAI_OP | FUNCTION | METH_CONTENT | WITH_CONTENT | PASS | STRING | LIST | DICT | VARIABLE | CONSTANT | NONE | COND_OPERATION | ASS_OPERATION

ITERATE_CONTENT -> FOR_CONTENT | WHILE_CONTENT | LIST | DICT | VARIABLE | CONSTANT

FLOW_CONTENT -> CONTINUE_CONTENT | BREAK_CONTENT

2. Conditional Grammar

IF -> a

ELIF -> b

ELSE -> c

IF_CONTENT -> IF_HEADER CONTENT | IF_CONTENT ELIF_CONTENT | IF_CONTENT ELSE_CONTENT

IF_HEADER -> IF COND_OPERATION COLON NEWLINE

ELIF_CONTENT -> ELIF_HEADER CONTENT | ELIF_CONTENT ELIF_CONTENT | ELIF_CONTENT ELSE_CONTENT

ELIF_HEADER -> ELIF COND_OPERATION COLON NEWLINE

ELSE_CONTENT -> ELSE_HEADER CONTENT

ELSE_HEADER -> ELSE COLON NEWLINE

3. Loop Grammar

3.1. For Loop

FOR -> d

IN -> e

FOR_CONTENT -> FOR_HEADER CONTENT

FOR_HEADER -> FOR FOR_VARIABLE IN ITERABLE COLON NEWLINE

FOR_VARIABLE -> VARIABLE | VARIABLE COMMA FOR_VARIABLE

ITERABLE -> STRING | LIST | VARIABLE | FUNCTION | METH_CONTENT | DICT

3.2. While Loop

WHILE -> f

WHILE_CONTENT -> WHILE_HEADER CONTENT

WHILE_HEADER -> WHILE COND_OPERATION COLON NEWLINE

3.3. Lain lain

ITERATE -> FOR_HEADER | FOR_CONTENT | WHILE_HEADER | WHILE_CONTENT

CONTINUE -> g NEWLINE

CONTINUE_CONTENT -> ITERATE CONTINUE | ITERATE CONTINUE CONTENT | CONTINUE

BREAK -> i NEWLINE

BREAK_CONTENT -> ITERATE BREAK | ITERATE BREAK CONTENT | BREAK

4. Pass Grammar

PASS -> h

5. Class Grammar

CLASS -> j

CLASS_CONTENT -> CLASS_HEADER CLASS_BODY

CLASS_HEADER -> CLASS NAME COLON NEWLINE | CLASS NAME NRM_BKT_OPN CLASS_PARAM NRM_BKT_CLS COLON NEWLINE

CLASS_PARAM -> CLASS_PARAM COMMA CLASS_PARAM | NAME

CLASS_BODY -> CLASS_BODY NEWLINE CLASS_BODY | CLASS_BODY NEWLINE | IF_CONTENT | ITERATE_CONTENT | CLASS_CONTENT | DEF_FUNC_CONTENT | ASS_OPERATION | FLOW_CONTENT | IMP_OPERATION | FUNCTION | METH_CONTENT | PASS | STRING

6. Def Function Grammar

6.1. Def function

DEF -> k

DEF_FUNC_CONTENT -> DEF_FUNC_HEADER DEF_FUNC_BODY

DEF_FUNC_HEADER -> DEF NAME NRM_BKT_OPN DEF_FUNC_PARAM NRM_BKT_CLS COLON NEWLINE | DEF NAME NRM_BKT_OPN NRM_BKT_CLS COLON NEWLINE

DEF_FUNC_BODY -> DEF_FUNC_BODY NEWLINE DEF_FUNC_BODY | CONTENT_FUNC

DEF_FUNC_RETURN -> RETURN ART_OPERATION | RETURN COND_OPERATION | RETURN DEF_FUNC_RETURN_VAL | RETURN

DEF_FUNC_RETURN_VAL -> METH_CONTENT | FUNCTION | VARIABLE | LIST | STRING | DICT | CONSTANT | NONE

DEF_FUNC_PARAM -> DEF_FUNC_PARAM COMMA DEF_FUNC_PARAM | NAME | NAME COLON NAME

RETURN -> l

6.2. Content

CONTENT_FUNC -> CONTENT_FUNC NEWLINE CONTENT_FUNC | CONTENT_FUNC NEWLINE | IF_CONTENT_FUNC | FOR_CONTENT_FUNC | WHILE_CONTENT_FUNC | CONTINUE_CONTENT_FUNC | BREAK_CONTENT_FUNC | CLASS_CONTENT | DEF_FUNC_CONTENT | ASS_OPERATION | IMP_OPERATION | RAI_OP | FUNCTION |

METH_CONTENT | WITH_CONTENT_FUNC | PASS | STRING | DEF_FUNC_RETURN | LIST | DICT | VARIABLE | CONSTANT | NONE

6.3. Conditional in function

IF_CONTENT_FUNC -> IF_HEADER CONTENT_FUNC | IF_CONTENT_FUNC ELIF_CONTENT_FUNC | IF_CONTENT_FUNC ELSE_CONTENT_FUNC

ELIF_CONTENT_FUNC -> ELIF_HEADER CONTENT_FUNC | ELIF_CONTENT_FUNC ELIF_CONTENT_FUNC | ELIF_CONTENT_FUNC ELSE_CONTENT_FUNC

ELSE_CONTENT_FUNC -> ELSE_HEADER CONTENT_FUNC

6.4. Loop in function

6.4.1. While Loop

WHILE_CONTENT_FUNC -> WHILE_HEADER CONTENT_FUNC

6.4.2. For Loop

FOR_CONTENT_FUNC -> FOR_HEADER CONTENT_FUNC

6.4.3. Continue and Break in function

ITERATE_FUNC -> FOR_HEADER | FOR_HEADER CONTENT_FUNC | WHILE_HEADER | WHILE_CONTENT_FUNC

CONTINUE_CONTENT_FUNC -> ITERATE_FUNC CONTINUE | ITERATE_FUNC CONTINUE CONTENT_FUNC | CONTINUE

BREAK_CONTENT_FUNC -> ITERATE_FUNC BREAK | ITERATE_FUNC BREAK CONTENT_FUNC | BREAK

6.4.4. With in function

WITH_CONTENT_FUNC -> WITH_HEADER CONTENT_FUNC

7. Conditional Operation Grammar

COND_OPERATION -> NRM_BKT_OPN COND_OPERATION NRM_BKT_CLS | COND_OPERAND COND_OPERATOR COND_OPERATION | COND_OPERAND

COND_OPERATOR -> REL_OP | LOG_OP | MEM_OP | IDN_OP

COND_OPERAND -> NOT COND_OPERAND | VARIABLE | CONSTANT | ART_OPERATION | METH_CONTENT | FUNCTION | LIST | STRING | NONE | NRM_BKT_OPN COND_OPERAND NRM_BKT_CLS

8. Assignment Operation Grammar

ASS_OPERATION -> VARIABLE ASS_OPERATOR ASS_OPERAND

ASS_OPERATOR -> ASS_OP

ASS_OPERAND -> NRM_BKT_OPN ASS_OPERAND NRM_BKT_CLS | VARIABLE | CONSTANT | COND_OPERATION | ART_OPERATION | METH_CONTENT | FUNCTION | LIST | STRING | DICT | NONE

9. Arithmetic Operation Grammar

ART_OPERATION -> NRM_BKT_OPN ART_OPERATION NRM_BKT_CLS | ART_OPERATION ART_OPERATOR ART_OPERATION | ART_OPERAND

ART_OPERATOR -> ART_OP | BIT_OP

ART_OPERAND -> VARIABLE | CONSTANT | METH_CONTENT | FUNCTION

10. Import Grammar

IMP_OPERATION -> FROM IMP_CONTENT IMPORT IMPT_MOD | IMPORT IMP_CONTENT | IMPORT AS_BLOCK

AS_BLOCK -> AS_BLOCK COMMA AS_BLOCK | IMP_CONTENT AS NAME

IMP_CONTENT -> IMP_CONTENT DOT IMP_CONTENT | NAME

IMPT_MOD -> NAME | ALL | AS_BLOCK | IMPT_MOD COMMA IMPT_MOD

ALL -> *

AS -> m

IMPORT -> n

FROM -> o

11. Raise Grammar

RAI_OP -> RAISE RAI_BODY

RAI_BODY -> NRM_BKT_OPN RAI_BODY NRM_BKT_CLS | VARIABLE | CONSTANT | COND_OPERATION | ART_OPERATION | METH_CONTENT | FUNCTION | LIST | STRING | DICT | NONE

RAISE -> p

12. And, Or, Not Grammar

AND -> q

OR -> r

NOT -> s

IS -> t

13. True False None Grammar

TRUE -> u

FALSE -> v

NONE -> w

14. Function and Method Grammar

14.1. Function

FUNCTION -> FUNCTION_BASE | FUNCTION_BASE VAR_IDX

FUNCTION_BASE -> VARIABLE NRM_BKT_OPN FUNCTION_PARAM NRM_BKT_CLS | VARIABLE NRM_BKT_OPN NRM_BKT_CLS

FUNCTION_PARAM -> FUNCTION_PARAM COMMA FUNCTION_PARAM | VARIABLE | CONSTANT | STRING | LIST | DICT | NONE | FUNCTION | METH_CONTENT | ART_OPERATION | COND_OPERATION | ASS_OPERATION

14.2. Method

METH_CONTENT -> METH_INIT DOT METH_CONTENT | METH_BACK_FUNC | METH_BACK_NAME

METH_CONTENT_NAME -> METH_INIT DOT METH_CONTENT_NAME | METH_BACK_NAME

METH_CONTENT_FUNC -> METH_INIT DOT METH_CONTENT_FUNC | METH_BACK_FUNC

METH_INIT -> NRM_BKT_OPN METH_INIT NRM_BKT_CLS | METH_BACK_FUNC | METH_BACK_NAME

METH_BACK_FUNC -> FUNCTION | FUNCTION VAR_IDX

METH_BACK_NAME -> NAME | NAME VAR_IDX

15. String, Dictionary, List Grammar

STRING -> z | z VAR_IDX | STRING MULTIPLY POSITIVE_NUMBER | STRING PLUS STRING

DICT -> CRL_BKT_OPN DICT_CONTENT CRL_BKT_CLS | CRL_BKT_OPN CRL_BKT_CLS | CRL_BKT_OPN NEWLINE DICT_CONTENT CRL_BKT_CLS | CRL_BKT_OPN NEWLINE DICT_CONTENT NEWLINE CRL_BKT_CLS | CRL_BKT_OPN DICT_CONTENT NEWLINE CRL_BKT_CLS

DICT_CONTENT -> DICT_TYPE COMMA DICT_TYPE | DICT_TYPE | DICT_TYPE COMMA NEWLINE DICT_TYPE

DICT_TYPE -> DICT_TYPES COLON DICT_TYPES

DICT_TYPES -> VARIABLE | CONSTANT | STRING | FUNCTION | METH_CONTENT | NONE

LIST -> SQR_BKT_OPN LIST_CONTENT SQR_BKT_CLS | SQR_BKT_OPN SQR_BKT_CLS

LIST_TYPE -> VARIABLE | CONSTANT | STRING | FUNCTION | METH_CONTENT | LIST | DICT | NONE

LIST_CONTENT -> LIST_TYPE | LIST_CONTENT COMMA LIST_TYPE | LIST_TYPE FOR VARIABLE IN FUNCTION

16. With Grammar

WITH_CONTENT -> WITH_HEADER CONTENT

WITH_HEADER -> WITH WITH_ST AS VARIABLE COLON NEWLINE

WITH_ST -> FUNCTION | METH_CONTENT_FUNC

WITH -> A

3.1.1.2. FA

1. Variable

VARIABLE -> NRM_BKT_OPN VARIABLE NRM_BKT_CLS | VAR_CTN

VAR_CTN -> NAME | NAME VAR_IDX | METH_CONTENT_NAME | METH_CONTENT_NAME VAR_IDX

VAR_IDX -> VAR_IDX VAR_IDX | [IDX] | [COLON] | [IDX COLON] | [COLON IDX] | [IDX COLON IDX] | [COLON COLON] | [COLON COLON IDX] | [COLON IDX COLON] | [COLON IDX COLON IDX] | [IDX COLON COLON] | [IDX COLON COLON IDX] | [IDX COLON IDX COLON] | [IDX COLON IDX COLON IDX]

IDX -> VARIABLE | CONSTANT | ART_OPERATION | STRING

2. Name

NAME -> x

3. Number

NUMBER -> PLUS NUMBER | MINUS NUMBER | NUMBER_CTN

POSITIVE_NUMBER -> PLUS POSITIVE_NUMBER | NUMBER_CTN

NUMBER_CTN -> y

CONSTANT -> NRM_BKT_OPN CONSTANT NRM_BKT_CLS | CON_CTN

CON_CTN -> TRUE | FALSE | NUMBER

4. Simbol

QUOTE -> '

QQUOTE -> "

COLON -> :

COMMA -> ,

DOT -> .

PLUS -> +

MINUS -> -

MULTIPLY -> *

NEWLINE -> NEWLINE NEWLINE | __new_line__ | __new_line2__

SPACE -> SPACE SPACE | __space__

OR_SYMBOL -> __or_symbol__

REL_OP -> = = | ! = | < = | > = | < | > | IS

ASS_OP -> = | + = | - = | * = | * * = | / = | / / = | % =

ART_OP -> + | - | * | * * | / | / / | %

LOG_OP -> AND | OR

MEM_OP -> IN | NOT SPACE IN

IDN_OP -> IS | IS SPACE NOT

BIT_OP -> & | OR_SYMBOL | ^ | > > | < < | ~\

SQR_BKT_OPN -> [

SQR_BKT_CLS ->]

CRL_BKT_OPN -> {

CRL_BKT_CLS -> }

NRM_BKT_OPN -> (

NRM_BKT_CLS ->)

3.1.2. Penjelasan sintaks python

1. Sintaks Conditional

Dapat dilihat pada CFG yang telah diimplementasikan bahwa untuk sintaks conditional dibuat beberapa aturan poroduksi.

Pertama untuk masing masing *if*, *elif*, dan *else* akan langsung meghasilkan simbol terminal yang berarti dia tidak meimiliki aturan penurunan lagi atau dapat dikatakan bahwa *if*, *elif*, dan *else* adalah basis dari aturan produksi untuk sintaks pada conditional.

Kedua, untuk masing masing if, elif, dan else kelompok kami membuat aturan produksi baru yaitu *if_header*, *if_content*, *elif_header*, *elif_content*, *else_header*, dan *else_content*. Masing masing dari aturan produksi tersebut memiliki sifat rekursif yang akan menghasilkan beberapa simbol non-terminal. Contoh nya adalah aturan produksi *if_header* akan menurunkan *if cond_operation colon newline*. Hasil penurunan dari aturan produksi *if_header* akan menurunkan simbol non-terminal yang akan menghasilkan simbol non-terminal lain atau simbol terminal.

2. Sintaks And, Or, Not, dan Is

Dapat dilihat pada CFG yang telah diimplementasikan bahwa untuk sintaks *and*, *or*, *not* dan *is* memiliki aturan produksi yang hasil penurunannya langsung menuju terminal simbol, yang berarti masing masing dari aturan produksi dari *and*, *or*, *not* dan *is* adalah aturan produksi yang merupakan basis.

3. Sintaks Looping

1. Sintaks While

Dapat dilihat pada CFG yang telah diimplementasikan bahwa untuk sintaks while dibuat beberapa aturan produksi.

Pertama, dibuat sebuah aturan produksi *while* yang langsung menghasilkan sebuah terminal simbol, ini menandakan bahwa aturan produksi *while* ini adalah basis untuk aturan produksi pada sintaks while.

Kedua, dibuat sebuah aturan produksi *while_header* dan *while_content* yang masing masing dari aturan produksi tersebut akan menghasilkan beberapa simbol non-terminal. Selain itu masing masing dari aturan produksi tersebut juga bersifat rekursif. Contoh nya adalah *while_content* yang akan menurunkan *while_header content*. Hasil penurunan dari aturan *while_content* akan menurunkan simbol simbol non-terminal atau simbol terminal lainnya.

2. Sintaks For

Dari CFG yang telah dibuat, terdapat aturan produksi *for* dan *in* menurunkan terminal simbol yang menandakan basis dari aturan produksi pada sintaks *for loop.* Untuk melakukan rekursifnya, terdapat aturan produksi *for_content*, *for_header*, *for_variable*, dan *iterable*. Pada *for_content*, bisa menurunkan non terminal *for_header* dan *content*, dimana content merupakan main contentnnya. Kemudian untuk *for_header* berperan sebagai headernya yaitu menurunkan *for, for_variable, in, iterable, colon,* dan *newline.* Sedangkan *for_variable* adalah aturan produksi yang menurunkan non terminal *variable*. Dimana kalau loopnya selesai berarti, *for _variable* menurunkan variable, jika masih looping maka *for_variable* menurunkan *variable, comma,* dan *for_variable* lagi.

3. Sintaks continue dan break

Dari CFG yang telah dibuat, untuk continue terdapat aturan produksi *continue* yang menurunkan terminal simbol *g* dan nonterminal simbol *newline*. Aturan ini berperan sebagai basis dari continue, dikarenkan menurunkan terminal simbol dan *newline.* Sementara itu, aturan produksi *continue_content* berfungsi sebagai rekurens yang bertugas untuk melanjutkan loopingnya, disebabkan oleh penurunan opsional dari :

1. *Iterate* dan *continue*
   Dalam hal ini *iterate* dan *continue* yang berperan sebagai non terminal symbol berfungsi untuk melanjutkan looping kembali, dengan kata lain Ketika melewati kedua nonterminal ini laju aturan produksi kembali ke atas lagi.
2. *Iterate*, *continue*, dan *content*
   Pada kasus ini sebenarnya hampir sama dengan sebelumnya, tetapi yang membuat berbeda adalah dengan adanya *content* sebagai nonterminal. Yang dalam hal ini, *content terdapat di* main content yang merupakan konten utama dari program, yang berakibat dia harus menjalankan konten-konten lain yang ada sebelum continue.
3. *Continue*
   Nonterminal ini merupakan kondisi akhir looping yang pada akhirnya akan diteruskan ke basis dari sintaks continue.

4. Sintaks Function

Pada sintaks function terdapat beberapa sintaks yang harus ada, yaitu sebagai berikut.

1. Sintaks Def

   Dari CFG yang telah dibuat, aturan produksi *def* dan return menurunkan terminal symbol yang menandakan basis dari aturan produksi pada sintaks *def*. Yang membedakan, pada def terjadi diawal sintaks, sedangkan return diproses diakhir sintaks (untuk mengakhiri fungsi). Selain itu, terdapat aturan produksi lain, yaitu *def_func_content*, *def_func_header*, *def_func_body, def_func_return, def_func_return_val,* dan *def_func_param*. Berikut ada uraian terkait aturan produksi tersebut.

   1. def_func_content

      Pada Nonterminal ini, diturunkan lagi Non terminal lain yaitu def_func_header dan def_func_body. Dimana pada bagian ini berperan untuk menjalankan content (isi) dari sintaks ini.

   2. def_func_header

      Pada Nonterminal ini, diturunkan lagi nonterminal :

      - *def, name, nrm_bkt_opn, def_func_param, nrm_bkt_cls, colon,* dan *newline.*
      - *def, name, nrm_bkt_opn, nrm_bkt_cls,* dan *colon.*

      Dimana nonterminal ini berperan sebagai header dari sintaks ini.

   3. def_func_body

      Pada Nonterminal ini, diturunkan lagi nonterminal:

      - *def_func_body, newline,* dan *def_func_body*
      - *content_func*

      Yang mana nonterminal ini berperan sebagai body dari sintaks ini.

   4. def_func_return

      Pada Nonterminal ini, diturunkan lagi nonterminal:

      - *return,* dan *art_operation*
      - *return,* dan *cond_operation*
      - *return,* dan *def_func_return_val*
      - *return*

      Yang mana pada nonterminal ini, berperan untuk proses untuk menghantarkan ke basisnya yang berakibat pada pemberhentian dari proses rekursifnya.

   5. def_func_return_val

      Pada nonterminal ini, diturunkan lagi nonterminal:

      - *meth_content*
      - *function*
      - *variable*
      - *list*
      - *string*
      - *dict*
      - *constant*

- *none*

Nonterminal ini berperan dalam proses pengembalian variable yang merupakan hasil dari fungsi tersebut.

6. def_func_param

Pada nonterminal ini, diturunkan lagi nonterminal:

- *def_func_param, comma, dan def_func_param*
- *name*
- *name, colon, name*

Nonterminal ini berperan mengindentifikasi parameter dari fungsi yang diberikan.

2. Sintaks While loop pada fungsi

Pada sintaks ini aturan produksi yang dibuat sama-sama menurunkan nonterminal *while_hedaer*, tetapi yang membuat berbeda adalah adanya penurunan ke nonterminal *content_func* yang merupakan rekurensi pada fungsi terkait dengan sintaks-sintaks lain yang ada di fungsi tersebut.

3. Sintaks For loop pada fungsi

Sama seperti While loop, pada sintaks ini aturan produksi yang dibuat hampir sama dengan aturan produksi for loop sebelumnya, tetapi yang membuat berbeda adalah adanya *for_content_func* yang merupakan rekurensi untuk proses looping pada fungsi terkait dengan sintaks-sintaks lain yang ada di fungsi tersebut.

4. Sintaks continue, break, dan with pada fungsi

Sebenarnya untuk sintaks, sama saja dengan poin sebelumnya. Pada continue dan break, terdapat penurunan nonterminal *continue_content_func* dan *break_content_func* yang berperan sebagai rekurens pada fungsi terkait dengan sintaks-sintaks lain yang ada di fungsi tersebut. Sementara nonterminal iterate_func berperan dalam proses looping didalam fungsi tersebut, baik dengan for loop maupun while loop. Sedangkan pada sintaks with juga sama, dengan *with_content_func* sebagai nonterminal yang menurunkan *with_header* dan *content_func*.

5. Sintaks True, False, dan None

Pada CFG yang telah dibuat Nonterminal true, false, dan none semuanya merupakan basis dari aturan produksi. Dikarenakan, pada aturan produksi tersebut, semua nonterminal true, false, dan none langsung menurunkan terminal.

6. Sintaks Import

Pada CFG yang telah dibuat, untuk aturan produksi pada sintaks terdapat beberapa nonterminal yang berperan pada proses pembangunan sintaks ini, antara lain:

1. imp_operation

Nonterminal ini berperan dalam proses inisiasi dari aturan produksi pada sintaks ini. Hal tersebut ditandai pada main content dengan adanya penurunan dari *start -> content -> imp_operation*. Selain itu, nonterminal ini juga berfungsi dalam proses rekurens pada sintaks ini.

2. *as_block*

   Nonterminal ini berfungsi untuk mengidentifikasi block pada sintaks ini yaitu dengan mengecek penulisan dari sintaks ini apakah sesuai atau tidak dengan yang diharapkan oleh grammar.

3. *imp_content*

   Nonterminal yang merupakan rekurensi dari sintaks ini berfungsi menjalankan content(isi) dari sintaks ini.

4. *impt_mod*

   Berfungsi sebagai rekurens yang merupakan perantara dari basisnya, yaitu ditandai dengan adanya penurunan :

   - *name*
   - *all*
   - *as_block*
   - *impt_mod, comma, impt_mod*

5. *all*

   Berfungsi sebagai basis (ditandai dengan penurunan terminal *)

6. *as*

   Berfungsi sebagai basis (ditandai dengan penurunan terminal m)

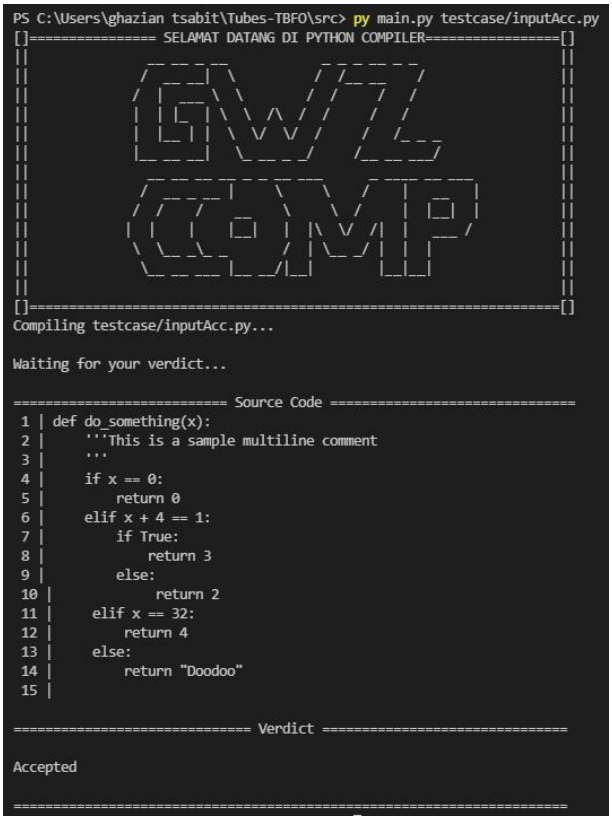
7. *import*

   Berfungsi sebagai basis (ditandai dengan penurunan terminal n)

8. *from*

   Berfungsi seabgai basis (ditandai dngan penurunan terminal o)

3.1.3. Tampilan Antarmuka



Gambar 3.1.2 Tampilan Antarmuka Prgoram Compiler Python 'GWZ COMP'

3.1.4. Fungsi dan Prosedur

converter.py

Fungsi	Deskripsi
function CFGfromTXT(grammarPath)	Fungsi ini bertujuan untuk mengubah grammar yang awalnya berada pada file txt menjadi sebuah grammar dalam bahasa python dalam bentuk dictionary
function displayGrammar(dict)	Fungsi iini bertujuan untuk menampilkan seluruh grammar yang ada ke layar
function isVariableValid(item)	Fungsi ini bertujuan untuk melakukan pengecekan apakah sebuah simbol pada

	grammar merupakan sebuah variabel atau simbol terminal pada sebuah CFG
function removeUnitProductioninCFG(CFG)	Fungsi ini bertujuan untuk menghilangkan produksi unit agar penurunan dari grammar yang telah dibuat menjadi lebih sederhana.
function turntoCNF(CFG)	Fungsi ini menghasilkan tepat satu simbol terminal atau 2 buah simbol non-terminal.
function CFGtoCNF(CFG)	Fungsi ini mengubah CFG menjadi CNF dengan memanfaatkan fungsi sebelumnya

CYK.py

Fungsi	Deskripsi
function CYK(input, CNF, src)	Fungsi ini bertujuan untuk mengimplementasikan CYK dari CNF yang telah dibuat

main.py

Fungsi	Deskripsi
function processInput(inp)	Fungsi ini bertujuan untuk mengubah source code yang telah di input ke dalam bentuk token
function fileReader(path)	Fungsi ini bertujuan untuk membuka dan membaca file yang diinput
function displayinp(inp)	Fungsi ini bertujuan mendisplay file yang telah dibaca
function startCompiler()	Fungsi ini menampilkan compiler GWZ

3.2. Hasil Pengujian

<table>
<tr><th>Source Code</th><th>Hasil Pengujian</th></tr>
<tr><td>inputAcc.py

```
def do_something(x):
    '''This is a sample multiline comment
    '''
    if x == 0:
        return 0
    elif x + 4 == 1:
        if True:
            return 3
        else:
            return 2
    elif x == 32:
        return 4
    else:
        return "Doodoo"
```

</td><td>

```
========================

Accepted

========================
```

</td></tr>
<tr><td>inputRejected.py

```
def do_something(x):
    '''This is a sample multiline comment
    '''
    x + 2 = 3
    if x == 0 + 1
        return 0
    elif x + 4 == 1:
        else:
            return 2
    elif x == 32:
        return 4
    else:
        return "Doodoo"
```

</td><td>

```
============================== Verdic

x + 2 = 3
^Syntax Error in line 4

=====================================
```

</td></tr>
<tr><td>test1.py</td><td>

```
============================

Accepted

============================
```

</td></tr>
</table>

```python
from math import __builtins__

def coba(x):
    s = 1
    T = [0 for i in range(x)]
    if (x == 10):
        print(10)
    elif (x == 3):
        print(3)
    else:
        x = 7
        T[3] = x

    for i in range(x):
        print(i)
        if (i == 6):
            s = "5"
            pass
    return s

y = 5
z = coba(y)
if (type(z) == str):
    raise TypeError
```

test2.py

```python
import math

def fungsi(x,z):
    y = math.sqrt(x)
    elif (type(z) == str):
        raise TypeError
    else:
        print(y)

fungsi(4,1)
fungsi(4,"Error")
```

```
============================== Verdict

elif (type(z) == str):
^Syntax Error in line 5

=======================================
```

test3.py

```
============================== Verdi

print("x"+x)
^Syntax Error in line 11

=======================================
```

```python
import numpy as np

def matriks(x):
    if (type(x) != int):
        raise TimeoutError
    else:
        i = 2
    x = np.add(x,x)
    y = np.transpose(x)
    for i in range(5):
        print("x"+x)
        if (i == 3):
            x = 3
            pass
        else:
            continue
```

test4.py

```python
Test = None
cek = True
if Test == None:
    cek = False
else:
    print("Tidak")
if cek == True:
    print("Hmmmmm")
else:
    print("Hore")
```

```
==========

Accepted

==========
```

test5.py

```python
from numpy.lib import scimath as smath

A = int(smath.log(1000))
B = int(smath.log(1500))
while(A >= 0 and B >= 0):
    A -= 1
    B -= 1
    if (A == 1):
        print("Woke")
        break
```

```
==========

Accepted

==========
```

test6.py

```python
class MyClass:
    for i in range(6):
        if i == 3:
            continue
        else:
            print("Gilang")
```

```
==========

Accepted

==========
```

test7.py

```python
x = 10
if  x not > 10:
    print("not returned True")
else:
    print("not returned False")
```

```
========== Verc

if  x not > 10:
^Syntax Error in line 2

==================================
```

test8.py

```python
x = ["apple", "banana", "cherry"]

y = x

a = x is y
print(a)

numbers = [1,2,3,4,5,6,7,8,9,10]

jumlah = 0
for i in range(len(numbers)):
    jumlah += numbers[i]

i = 0
jumlah1 = 0
while i < len(numbers):
    jumlah1 += numbers[i]
    i += 1

print(jumlah)
print(jumlah1)

x = 10
if not x > 10:
    print("not returned True")
else:
    print("not returned False")
```

```
==========

Accepted

==========
```

test9.py

```
from PIL import Image

var = "input.jpg"
ACCEPTED_ASCII_LIST = []

def extract_text_from_image():
    PIPE_ASCII = 124

    time_start = time.perf_counter()
    text = pytesseract.image_to_string(Image.open(IMAGE_INPUT_NAME))
    with open(FILE_OUTPUT_NAME, 'asdf') as f_out:
        for c in text:
            ascii = ord(c)
            if ascii == PIPE_ASCII:
                f_out.write('I')
            elif ascii in ACCEPTED_ASCII_LIST:
                f_out.write[c]
            else:
                f_out.write(' ')
    time_end = time.perf_counter()
```

```
==============================

Accepted

==============================
```

BAB IV

PEMBAGIAN TUGAS

NIM	Nama	Pembagian Tugas
13520137	Muhammad Gilang Ramadhan	Pembuatan grammar.txt, menulis laporan bab 2 (Teori Dasar), pembuatan program main.py .
13520160	Willy Wilsen	Pembuatan program pada file CYK.py, menulis laporan bab 3, pembuatan program main.py.
13520165	Ghazian Tsabit Alkamil	Pembuatan program yang terdapat pada file converter.py, menulis laporan bab 5, pembuatan program main.py.

BAB V

KESIMPULAN, SARAN , REFLEKSI

5.1. Kesimpulan

Berdasarkan implementasi dan hasil pengujian yang telah dilakukan maka didapatkan kesimpulan sebagai berikut :

1. Program *compiler* python dengan menerapkan Algoritma CYK dapat dijalankan dengan baik dan sesuai dengan spesifikasi yang telah ditentukan.
2. Perlu dibuat sebuah *context free grammar syntax* python dalam pembuatan program compiler python.
3. Konsekuensi dari penggunaan algoritma CYK adalah perlu dibuat sebuah program yang mengubah *context free grammar syntax* python ke bentuk *chomsky normal form.*
4. Implementasi dari program *compiler* python dapat berjalan baik dengan bantuan program yang berfungsi untuk mengubah *source code* yang diujikan ke bentuk token.
5. Beberapa *test case* membutuhkan waktu yang cukup besar untuk mengeluarkan hasil evaluasi *syntax* dan variable.

5.2. Saran

Berdasarkan impelementasi program dan hasil pengujian yang telah dilakukan, terdapat beberapa saran dari penulis sebagai berikut:

1. Gunakan Algoritma CYK pada implementasi program compiler karena dokumentasi yang dapat dijadikan referensi untuk algoritma ini sudah sangat banyak.
2. Implementasikan algoritma yang lebih cepat dan efisien agar waktu yang dibutuhkan untuk mengevaluasi *source code* tidak terlalu besar.

5.3. Refleksi

Dalam proses pengerjaan tugas besar ini tentu kami menemukan beberapa kendala, namun dari kendala kendala tersebutlah kami banyak belajar. Hal-hal yang menjadi refleksi pada saat kami mengerjakan tugas besar ini adalah sebagai berikut:

1. Pemahaman yang kuat tentang *context free grammar* dan *chomsky normal form* sangat dibutuhkan pada saat pengerjaan program ini.
2. Diperlukan tingkat ketelitian yang tinggi pada saat pembuatan *context free grammar* untuk syntax – syntax python.

REFERENSI

"Cocke-Younger-Kasami (CYK) Algorithm"

https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/

"Full Grammar Specification"

https://docs.python.org/3/reference/grammar.html

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). Context Free Grammars and Languages. In *Introduction to automata theory, languages, and computation*. essay, Pearson/Addison Wesley.