LAPORAN TUGAS BESAR COMPILER BAHASA PYTHON

IF2124/Teori Bahasa Formal dan Automata

Dipersiapkan oleh:

Agen Parser

1. Nayotama Pradipta - 13520089
2. Angelica Winasta Sinisuka - 13520097
3. Muhammad Rakha Athaya - 13520108

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

BAB I

Teori Dasar

1.1 Finite Automata

Secara sederhana, *finite automata* dapat diartikan sebagai suatu model komputasional abstrak dan sederhana yang memiliki lima elemen atau *tuples*. Elemen dari sebuah *finite automata* meliputi:

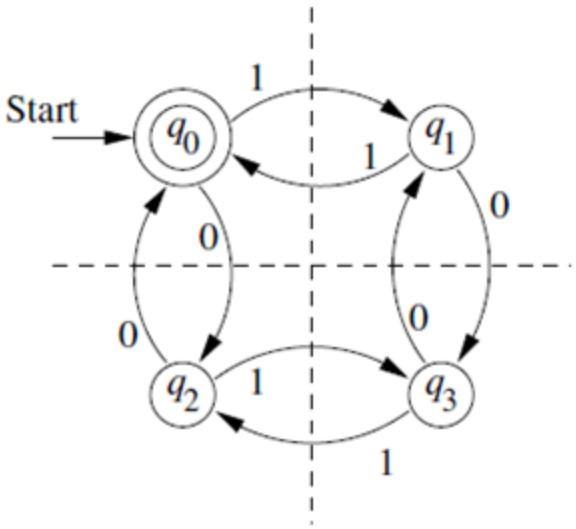
1. *Q : Finite set of states*
2. *Σ : Set of Input symbols*
3. *q : Initial state*
4. *F : Set of final states*
5. δ *: Transition function*

Ciri khas dari *finite automata* adalah adanya transition rules yang mentransformasikan dari satu state ke state lain. Representasi FA paling sederhana adalah menggunakan graf ataupun tabular. Di dalam graf, *nodes* menandakan *states* atau keadaan, panah menunjukkan transisi, serta label pada panah menunjukkan input yang menyebabkan transisi. Pada representasi tabular, tabel diisi dengan fungsi δ yang menunjukkan *set of states* dan input symbols.

Transition function (δ) menerima dua argumen, yaitu sebuah *state* dan sebuah input symbol. Sebagai contoh:

δ(*q,a*) menunjuk kepada *state* yang dituju ketika FA berada pada state *q* dan menerima input dalam bentuk *a*.

FA dibagi menjadi dua bentuk/tipe, yaitu DFA (*Deterministic Finite Automata)* dan NFA (*Non-deterministic Finite Automata*). Pada DFA, automata hanya dapat bergerak ke satu state ketika menerima satu input dan tidak bisa menerima null (ε). Pada DFA juga dikenal istilah *dead state*, yaitu state yang sudah tidak mungkin menuju ke final state. Suatu DFA dapat didefinisikan dalam *five-tuple notation*: $A = (Q, \Sigma, \delta, q_0, F)$. Pada DFA, fungsi transisi didefinisikan sebagai δ: $Q \times \Sigma \rightarrow Q$.



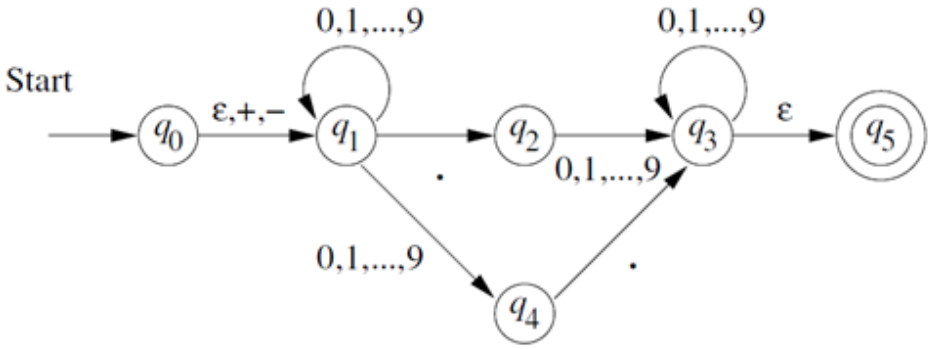
Gambar 1.1.1 Graf/Diagram DFA yang menerima input berupa 0 dan 1 dengan masing-masing berjumlah yang genap

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Gambar 1.1.2 Representasi Tabular untuk DFA yang sama pada gambar 1.1.1

Pada NFA, automata dapat pindah menuju lebih dari satu state ketika menerima satu input. Selain itu, terdapat tipe NFA bernama ϵ-NFA yang juga memungkinkan *transition* ke state lain hanya dengan input ϵ atau *empty string*.

Pada NFA, fungsi transisi didefinisikan sebagai δ: *Q X (Σ U* ϵ*)* → *2 ^ Q* .



Gambar 1.1.3 Representasi graf ϵ-NFA yang menerima input berupa bilangan desimal

	ϵ	$+, -$	.	$0, 1, \ldots, 9$
q_0	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Gambar 1.1.4 Representasi tabular ϵ-NFA yang sama dengan gambar 1.1.3

Pada fungsi transisi kedua bentuk FA, *Q* adalah subset dari 2^Q, maka dapat disimpulkan bahwa semua DFA adalah NFA tetapi tidak berlaku sebaliknya. Meskipun demikian, NFA dapat diubah menjadi DFA yang ekivalen. Jika ditinjau dari fungsinya, NFA lebih digunakan untuk konsep dan teori, sedangkan DFA digunakan pada compiler di bagian lexical analysis.

1.2 Context Free Grammar

Context free grammar merupakan notasi untuk *languages* yang lebih kuat dibandingkan FA (finite automata) atau RE(*regular expression*), tetapi tidak mendefinisikan semua *languages* yang mungkin. Context free grammar sangat baik digunakan untuk struktur yang *nested*, seperti tanda kurung di bahasa pemrograman

Terdapat 3 komponen didalam CFG yaitu:
1. Terminal: Simbol dari huruf language yang didefinisikan.

2. Variables: Variabel disebut juga dengan non terminal, yaitu set simbol yang berhingga dan setiap set melambangkan suatu bahasa.

3. Start Symbol: variabel yang bahasanya sedang didefinisikan

Produksi dari CFG punya bentuk variabel, yaitu string yang terdiri dari variabel dan terminal. Konvensinya adalah sebagai berikut:

- A,B,C, …. : variabel
- a, b,c, ….. : terminal
- ….,.X,Y,X : terminal atau variabel
- …..,q,x,y,z : terminal strings

Derivation dilakukan pada string language CFG pada start simbol. Kemudian menggantikan beberapa variabel A ke kanan dari produksinya. I*terated Derivation* ialah langkah yang melakukan derivation nol kali atau lebih dari nol. *Sentential Forms* merupakan string yang variabel atau/dan terminal *derived* dari start simbol.

Jika G merupakan CFG, maka L(G) yang merupakan language dari G adalah {w| S => *w} dengan syarat w harus merupakan string terminal dan S adalah start simbol. Derivation memperbolehkan untuk mengganti variabel dalam string. Terdapat banyak cara saat derivaion dengan derivation string. Kita dapat menggantikan variabel paling kanan atau kiri untuk menghindari perbedaan.

Contoh dari *leftmost derivation* dan *rightmost derivation* ialah *balanced-parentheses grammar* ,sebagai berikut:

S → SS | (S) | ()

S → SS → (S)S → (())S → (())()

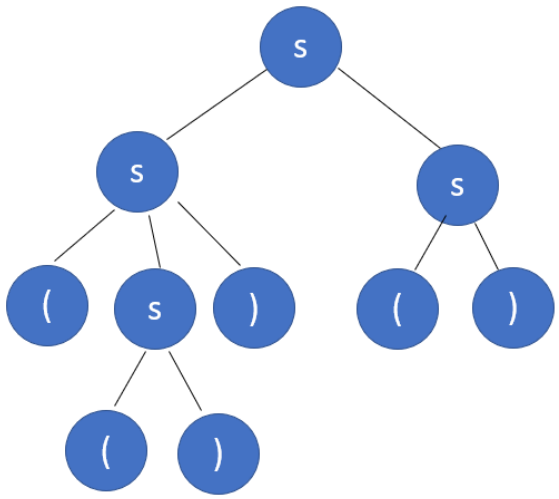
sehingga , S→ * (())()

Contoh untuk Rightmost Derivation

S → SS |(S) | ()

S → SS → S() → (S)() → (())()

Parse trees merupakan pohon yang digambarkan oleh simbol dari CFG yang spesifik. Daun (*leaves*) digambarkan dengan terminal atau epsilon. *Interior nodes* digambarkan dengan variabel, dan *root* digambarkan oleh *start symbol.* Contoh dari *parse tree* adalah sebagai berikut:



Gambar 1.2.1 parse tree

Concatenation dari label daun dari urutan kiri ke kanan *(preorder transversal)* disebut dengan *yield* dari parse tree. *Yield* yang dihasilkan dari parse tree di atas adalah (())()

Untuk setiap *parse tree*, terdapat *unique leftmost* dan *unique rightmost derivation*. CFG dikatakan ambigu apabila terdapat string dalam *language* yang menghasilkan 2 atau lebih parse trees. Sebagai contoh, ekspresi di prolang. Operator + dan * dan argumen merupakan identifiers, i.e strings di

L((a + b)(a+b+0+1)*)

Ekspresi tersebut didefinisikan oleh grammar

G= ({E,I},T,P,E)

T = {+,*,(,),a,b,O,1} dan P adalah set produksi sebagai berikut

1. E → I
2. 2. E → E +E
3. 3. E → E * E
4. E → (E)
5. I → a
6. I → b
7. I → Ia

8. I → Ib
9. I → IO
10. I → I1

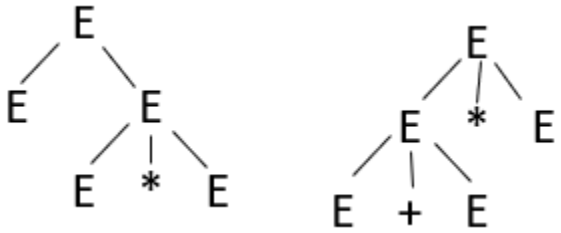
Sentential form E + E * E mempunyai 2 *derivation*, yaitu

$$E \rightarrow E + E \rightarrow E + E * E$$

dan

$$E \rightarrow E * E \rightarrow E + E * E$$

Yang memberikan kita 2 bentuk *parse trees*, yaitu



Gambar 1.2.2 2 buah parse tree

1.3 Python Syntax

Bahasa pemrograman Python adalah bahasa yang dieksekusi oleh sebuah interpreter. Interpreter tersebut bertugas untuk memparsing sintaks python, dan kemudian merubahnya menjadi sebuah instruksi mesin baris per baris. Terdapat beberapa aturan penulisan sintaks pada bahasa pemrograman Python. Aturan-aturan tersebut harus diikuti agar interpreter bisa memparsing dan menjalankan aplikasi dengan baik. Jika tidak, maka aplikasi tidak akan berjalan dan memproduksi sebuah error. Secara umum, sintaks penulisan python bersifat:

- *Case sensitive*
- Tidak menggunakan titik koma
- Indentasi digunakan sebagai pembentuk struktur
- Tidak ketat terhadap tipe data
- *Human friendly*

Case sensitive, artinya dalam bahasa Python huruf kecil dan besar (kapital) akan dianggap sebagai dua hal yang berbeda. Sebagai contoh, “kota_tua” ≠ “Kota_Tua”. *Statement* adalah sebuah pernyataan atau instruksi yang akan dieksekusi oleh mesin. Pada umumnya di bahasa pemrograman lain setiap statement akan dibedakan berdasarkan adanya karakter titik koma (;), berbeda dengan Python yang ***menggunakan karakter ganti baris*** (\n). Setiap pergantian baris, interpreter akan menganggap bahwa sebuah statement telah sempurna.

Namun, karakter titik koma tetap dapat digunakan apabila terdapat banyak statement dalam satu baris.

Dalam python, ***indentasi digunakan untuk mendefinisikan struktur blok kode*** program. Kesalahan indentasi bisa berujung pada sebuah error. Indentasi itu sendiri adalah penulisan paragraf yang agak menjorok masuk ke dalam. Dalam penulisan kode, indentasi dapat dibentuk dengan menggunakan spasi beberapa kali atau tab.

Selain itu, bahasa python bersifat ***tidak ketat terhadap tipe data***, di mana kita bisa memberi dan mengubah nilai apapun dari tipe data apapun ke dalam sebuah variabel. Sebagai contoh, variabel ***a*** yang awal mula diisi dengan nilai integer *100* dapat kita ganti isinya dengan nilai string “kesukaanku”.

BAB II

Hasil Finite Automata dan Context Free Grammar

2.1 Isi Context Free Grammar

S → S S	EXP → EXP B
S → COM IM	B → BINARY_OP EXP
S → IM COM	B → ARITHMETIC_OP EXP
S → METHOD COM	EXP → LP C
S → COM METHOD	C → EXP RP
S → COM	COMP_OP → 'DOUBLEEQUAL'
S → IM	COMP_OP → 'GREATER_OR_EQUAL_THAN'
S → METHOD	COMP_OP → 'LESS_OR_EQUAL_THAN'
S → FOR_METHOD	COMP_OP → 'GREATER_THAN'
S → PRINT_METHOD	COMP_OP → 'LESS_THAN'
S → IFELSE	COMP_OP → 'NOT_EQUAL'
S → WHILE_METHOD	BINARY_OP → 'AND'
S → DEF	BINARY_OP → 'OR'
S → WITH	NOT → 'NOT'
S → CLASS	WHILE_METHOD → WHILE W1
S → OBJ	W1 → IDENTIFIER W4
S → ASSIGN	W1 → EXP W4
S → RETURN	W1 → LP W2
S → PASS	W2 → IDENTIFIER W3
S → RAISE	W2 → EXP W3
S → BREAK	W3 → RP W4
S → CONTINUE	W4 → COLON NEWLINE
S → ARBITRARY_METHOD	W4 → COLON
COM → COMS NEWLINE	WHILE → 'WHILE'
COM → COMS	NEWLINE → 'NEWLINE'
COMS → 'COMMENT'	DEF → DEFW DEF1
IM → 'FROM' A	DEF1 → IDENTIFIER DEF2
IM → B	DEF1 → IDENTIFIER COLON
A → IDENTIFIER B	DEF2 → DEF4
B → 'IMPORT' C	DEF2 → LP DEF3
C → IDENTIFIER D	DEF3 → DEF4
C → IDENTIFIER	DEF3 → IDENTIFIER DEF4
D → 'AS' E	DEF3 → STRING DEF4
E → IDENTIFIER IMEND	DEF3 → NUMBER DEF4
E → IDENTIFIER	DEF3 → IDENTIFIER
IMEND → NEWLINE	DEF3 → STRING
IDENTIFIER → 'IDENTIFIER'	DEF3 → NUMBER
WITH_METHOD	DEF3 → DEF3 DEFINBETWEEN
IDENTIFIER → 'IDENTIFIER'	DEFINBETWEEN → COMA DEF3
IDENTIFIER → NUMBER	DEF4 → RP COLON
IDENTIFIER → 'TRUE'	

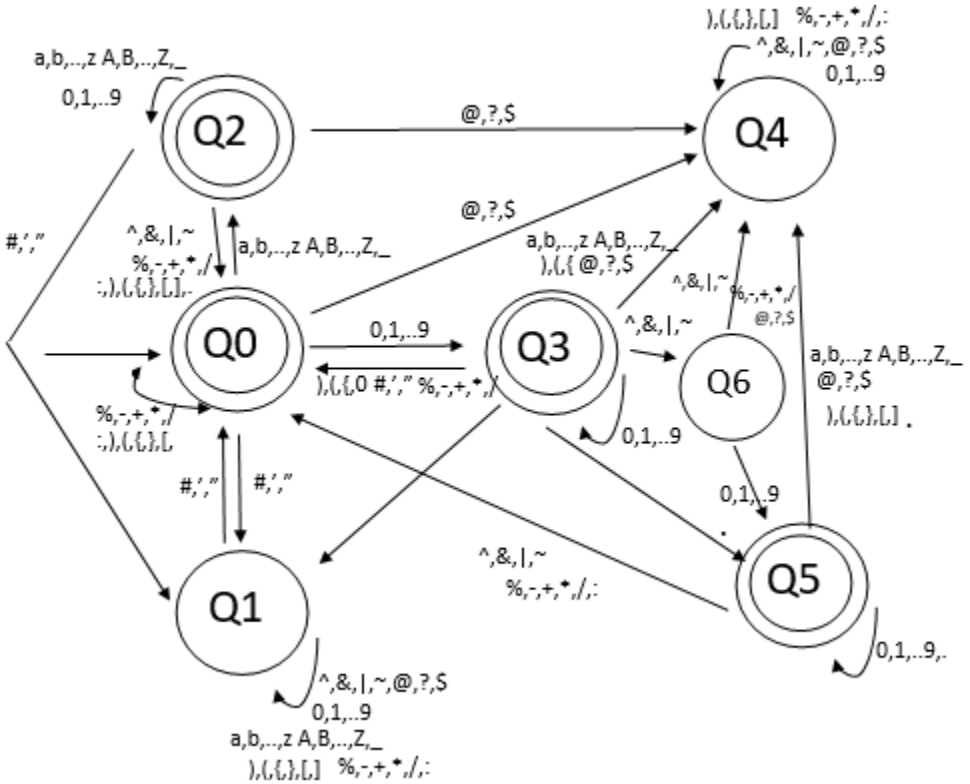
IDENTIFIER → 'FALSE'
IDENTIFIER → OBJ
WITH_METHOD → 'WITH_METHOD'
IDENTIFIER
PRINT_METHOD → PRINT M1
M1 → LP M2
M2 → OBJ RP
M2 → RP
M2 → STRING MS
M2 → FLOAT MF
M2 → IDENTIFIER MS
M2 → NUMBER
M2 → EXP RP
MS → RP
MS → PLUS MPLUSSTRING
MS → MULTIPLY MMULTIPLYINTEGER
MS → COMA M2
MMULTIPLYINTEGER → NUMBER MS
MPLUSSTRING → STRING MS
MPLUSSTRING → IDENTIFIER MS
MF → RP
MF → COMA M2
MF → PLUS MNUMBER
MF → MULTIPLY MNUMBER
MF → MINUS MNUMBER
MF → DIVIDE MNUMBER
MF → POWER MNUMBER
MI → RP
MI → COMA M2
MI → PLUS MNUMBER
MI → MULTIPLY M2
MI → MINUS MNUMBER
MI → DIVIDE MNUMBER
MI → POWER MNUMBER
MNUMBER → NUMBER MF
MNUMBER → NUMBER MI
MNUMBER → FLOAT MF
MNUMBER → FLOAT MI
NUMBER → 'NUMBER'
MULTIPLY → 'MULTIPLY'
PRINT → 'PRINT'
PLUS → 'PLUS'
COMA → 'COMA'
FUNC → 'FUNC'
STRING → 'STRING'
FLOAT → 'FLOAT'
FOR_METHOD → FOR F1
F1 → IDENTIFIER F2
F2 → IN F3
F3 → IDENTIFIER COLON
DEFW → 'DEF'
WITH → WITHW WITH0
WITH0 → IDENTIFIER WITH1
WITH1 → LP WITH2
WITH2 → IDENTIFIER WITH3
WITH2 → STRING WITH3
WITH2 → NUMBER WITH3
WITH3 → RP WITH4
WITH3 → COMA WITH2
WITH4 → COLON
WITH4 → 'AS' WITH5
WITH5 → IDENTIFIER COLON
WITHW → 'WITH'
CLASS → CLASSW CLASS1
CLASS1 → IDENTIFIER COLON
CLASS1 → IDENTIFIER DEF2
CLASSW → 'CLASS'
OBJ → 'STRING'
OBJ → NUMBER
OBJ → 'IDENTIFIER'
OBJ → 'FALSE'
OBJ → 'NONE'
OBJ → 'TRUE'
OBJ → OBJ OBJ1
OBJ → STRING OBJ1
OBJ → OBJ OBJ3
OBJ → OBJ OBJ0
OBJ → OBJ OBJ0
OBJ → OBJ OBJIN
OBJIN → IN OBJ
OBJ0 → ARRAY
OBJ0 → ARRAY OBJ1
OBJ0 → ARRAY OBJ3
OBJ1 → 'WITH_METHOD' OBJ2
OBJ2 → OBJ
OBJ2 → 'IDENTIFIER' ARRAY
OBJ2 → OBJ OBJ3
OBJ3 → LP OBJ4
OBJ3 → LP OBJRP
OBJRP → RP
OBJ4 → IDENTIFIER OBJ5
OBJ4 → STRING OBJ5
OBJ4 → NUMBER OBJ5
OBJ4 → OBJ OBJ5
OBJ4 → EXP OBJ5
OBJ4 → OBJ5
OBJ5 → ARITHMETIC_OP AOP1
AOP1 → OBJ4
OBJ5 → RP
OBJ5 → RP OBJ1

F3 → ARBITRARY_METHOD F10
F10 → RP COLON
F10 → RP
FOR → 'FOR'
IN → 'IN'
LEN → 'LEN'
RANGE → 'RANGE'
COLON → 'COLON'
IFELSE → IF IF2
IFELSE → IF4
IFELSE → IF7
IFELSE → IF IF97
IF97 → OBJ IF0
IF0 → IN IF1
IF1 → OBJ COLON
IF2 → EXP IF3
IF3 → COLON IFNL
IF3 → COLON
IFNL → NEWLINE IFNEXT
IFNEXT → CONTENT
IF4 → ELIFTOK IF99
IF99 → ELIF IF5
IF5 → EXP IF6
IF5 → IDENTIFIER IF0
IF6 → COLON IFNL
IF6 → COLON
IF7 → ELIFTOK IF98
IF98 → ELSE IF8
IF8 → COLON IF9
IF8 → COLON
IF9 → NEWLINE IF10
IF9 → IF10
IF10 → CONTENT
IF → 'IF'
ELSE → 'ELSE'
ELIF → 'ELIF'
ELIFTOK → 'ELIFTOK'
CONTENT → S
EXP → OBJ
EXP → OBJ A
EXP → EXP A
A → COMP_OP EXP
EXP → NOT EXP
EXP → NOT EXP

OBJ2 → OBJ1
ASSIGN → OBJ ASS1
ASS1 → EQUALS OBJ
ASS1 → EQUALS NUMBER
ASS1 → EQUALS STRING
ASS1 → EQUALS ARRAY
ASS1 → EQUALS EXP
ARRAY → 'LB' ARR1
ARR1 → 'RB'
ARR1 → OBJ 'RB'
ARR1 → STRING 'RB'
ARR1 → NUMBER 'RB'
ARR1 → 'IDENTIFIER' ARR2
ARR1 → 'IDENTIFIER' 'RB'
ARR2 → FOR_METHOD 'RB'
EQUALS → 'EQUALS'
ARITHMETIC_OP → 'MINUS'
ARITHMETIC_OP → 'PLUS'
ARITHMETIC_OP → 'MULTIPLY'
ARITHMETIC_OP → 'DIVIDE'
ARITHMETIC_OP → 'POWER'
ARITHMETIC_OP → 'MOD'
ARITHMETIC_OP → 'MINUS' EQUALS
ARITHMETIC_OP → 'PLUS' EQUALS
ARITHMETIC_OP → 'MULTIPLY' EQUALS
ARITHMETIC_OP → 'DIVIDE' EQUALS
ARITHMETIC_OP → 'POWER' EQUALS
ARITHMETIC_OP → 'MOD' EQUALS
RETURN → 'RETURN' OBJ
PASS → 'PASS'
RAISE → 'RAISE' OBJ
BREAK → 'BREAK'
CONTINUE → 'CONTINUE'
ARBITRARY_METHOD → IDENTIFIER
ARM1
ARM1 → LP ARM2
ARM2 → ARBITRARY_METHOD RP
ARM2 → ARM1 RP
ARM2 → OBJ RP
ARM2 → EXP RP
LP → 'LP'
RP → 'RP'
MULTILINE → "TRIPLEQUOTE"

2.2 Isi Finite Automata

Penggunaan finite automata digunakan untuk pengecekan jika variabel valid atau tidak. Kami menggunakan asumsi bahwa variabel akan dicek setelah atau sebelum operator aritmatika, assignment, perbandingan, logical, bitwise, serta variabel dalam *parentheses* dan *brackets*. Berikut merupakan gambaran diagram DFA



Gambar 2.2.1 Graf/Diagram DFA yang menerima input berupa variabel sesuai dengan sintaks python

BAB III

Implementasi dan Pengujian

3.1 Implementasi

Garis Besar

Program yang kami buat terdiri atas enam file utama dan beberapa file input untuk pengetesan. Dari enam file utama, lima merupakan file python sedangkan yang satu adalah text file yang berisi context free grammar. Pada dasarnya file yang berisi alur program ada pada main.py. File ini memerlukan import dari file lexer, parser, dan lexer Rules. Program main akan meminta file input dalam bentuk python file ataupun text file. Kemudian program akan membuat objek dengan fungsi lexer, yang menerima rules dari lexerRules dan digunakan untuk proses lexing. Setelah lexing, program akan melakukan parsing dengan bantuan parser.py serta input grammar.txt. Pada saat proses parsing, program akan menghitung jumlah syntax error yang ditemukan hingga keseluruhan input file telah diparse. Jika tidak ada error maka program akan output ke terminal berupa “Accepted”, jika ada error maka program akan memunculkan jumlah error serta menampilkan kalimat yang menyebabkan error. Program juga dapat menghitung run time menggunakan module time.

File

1. grammar.txt

File ini berisi context free grammar yang akan digunakan untuk mengevaluasi syntax file input.

2. cfgtocnf.py

File ini digunakan untuk meng*convert* CFG pada grammar.txt menjadi sebuah grammar dalam bentuk Chomsky Normal Form. Di dalam file ini terdapat tiga fungsi, yaitu:

- Read_grammar : Membaca grammar dari sebuah file
- Add_rule : Menambahkan rule ke dalam RULE_DICT yang merupakan global var
- convert_grammar: Fungsi utama pada file ini yaitu untuk mengubah grammar pada grammar.txt yang berbentuk CFG menjadi CNF

3. CYKparser.py

Seperti namanya, file ini berisi parser yang mengimplementasikan algoritma Cocke-Younger-Kasami (CYK). Tujuan dari adanya file cfgtocnf.py adalah karena algoritma CYK membutuhkan grammar CNF sebagai masukannya. File ini berisi dua class, yaitu sebuah Node dan Parser itu sendiri. Di dalam class Node terdapat konstruktor init untuk mengkonstruksi symbol, anak1, dan anak2, serta terdapat method repr untuk merepresentasikan objek sebagai string. Di dalam class Parser terdapat konstruktor init untuk mengkonstruksi tabel parsing, production rules, grammar, string input, dan sentence. Di dalam class Parser juga ada method call dan dell, serta fungsi untuk membaca dari file maupun dari string. Ada fungsi parse untuk melakukan parsing pada hasil input file, dan yang terakhir ada fungsi print tree untuk menghasilkan output berupa kesalahan syntax jika ada kesalahan.

4. testdfa.py

File ini mendeklarasikan states-states yang dibutuhkan untuk menerima input berbagai simbol pada file eksternal. File ini mengandung fungsi dfa yang akan memproses dan memilah kata yang termasuk variabel tiap baris dalam file eksternal.

5. Lexer.py

File ini berfungsi untuk mengubah input menjadi token-token yang akan digunakan pada proses parsing. File ini memiliki satu class bernama Changing, dan di dalamnya terdapat konstruktor, fungsi token, dan fungsi tokens.

6. lexerRules.py

File ini isinya rules yang digunakan pada lexer.py. Aturan ini termasuk triple quotes, number and floats, mathematical operators, punctuation, comparison operators, value assignment, whitespace and newline, string, print, identifier, random case, serta python keyword yang harus terdaftar sesuai dengan spek tubes.

7. Main.py

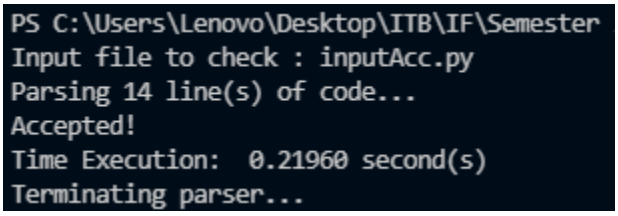
File ini merupakan program utama yang dijalankan untuk mengevaluasi sintaks dan nama-nama variabel dari input file eksternal yang berisi string kode program Python. Jika input diterima akan meng-*output* “Accepted!” atau “Syntax Error!” (beserta jumlah error-nya) bila tidak diterima.

3.2 Pengujian

File inputAcc.py

Gambar 3.2.1 Program inputAcc.py

Hasil pengujian



Gambar 3.2.2 Output proses inputAcc.py

Penjelasan

File inputAcc tidak memiliki syntax yang bermasalah ataupun tidak sesuai dengan grammar sehingga file diterima oleh program (output dalam bentuk “Accepted!”).

File inputReject.py

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

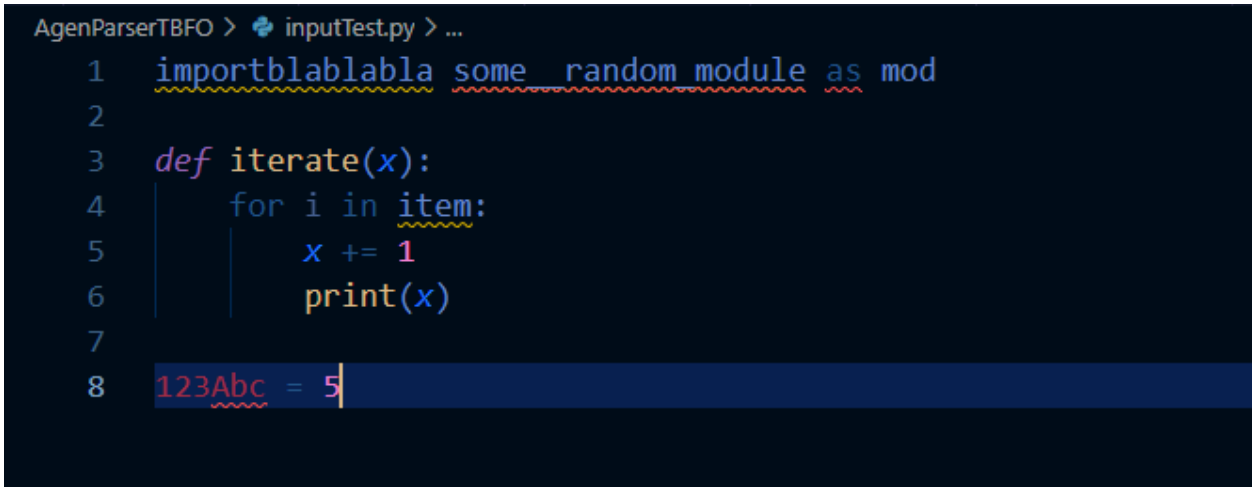
Hasil pengujian

```
FO/main.py"
Input file to check : inputReject.py
Parsing 14 line(s) of code...
Kalimat tidak termuat dalam bahasa dari grammar yang diberikan!
 IDENTIFIER PLUS NUMBER EQUALS NUMBER
Kalimat tidak termuat dalam bahasa dari grammar yang diberikan!
  IF IDENTIFIER DOUBLEEQUAL NUMBER PLUS NUMBER
Kalimat tidak termuat dalam bahasa dari grammar yang diberikan!
 ELIF IDENTIFIER DOUBLEEQUAL NUMBER COLON
Kalimat tidak termuat dalam bahasa dari grammar yang diberikan!
 ELSE COLON
Syntax Error! 4 error yang ditemukan dalam file.
Time Execution:  0.34701 second(s)
Terminating parser...
```

Penjelasan

File inputReject memiliki beberapa syntax yang bermasalah dan tidak sesuai dengan grammar, sehingga file ditolak oleh program (output dalam bentuk “Syntax Error!”). Error pertama dari baris keempat dimana “x + 2 = 3” berbentuk identifier plus number equals number yang tidak ada dalam bahasa dari grammar (karena “=” untuk assignment). Pada baris kelima, pernyataan if semestinya diakhiri dengan kolon. Pada baris kesepuluh, pernyataan elif didahului pernyataan else yang menjadi pernyataan pertama dalam bloknya. Pada baris kedua belas salah karena tidak terdapat dalam bahasa dari grammar.

File inputTest.py



Hasil pengujian

```
Input file to check : inputTest.py
Parsing 8 line(s) of code...
Kalimat tidak termuat dalam bahasa dari grammar yang diberikan!
IDENTIFIER IDENTIFIER AS IDENTIFIER
123Abc = 5 Rejected
Syntax Error! 2 error yang ditemukan dalam file.
Time Execution:  0.15259 second(s)
Terminating parser...
```

Penjelasan

File inputTest memiliki dua kesalahan yaitu pada baris 1 dan baris 8. Pada baris 1, “importblablabla” diidentifikasi sebagai identifier. Hal ini dikarenakan “importblablabla” bukan merupakan python keyword “import”. Ketika program membaca baris pertama tersebut, maka language yang terbentuk adalah identifier identifier AS identifier. Hal ini bukan termasuk grammar yang benar. Oleh karena itu, program menganggap baris satu sebagai baris dengan syntax error. Selanjutnya pada baris 8 terdapat variabel bernama 123Abc yang bernilai 5. Sebenarnya baris 8 termasuk dalam grammar yang sesuai, akan tetapi penamaan variabel yang dimulai dengan angka memicu DFA dalam deteksi variabel error. Kesimpulannya terdapat dua jenis syntax error pada file inputTest.py, yaitu grammar error dan variabel error.

BAB IV

Kesimpulan dan Saran

4.1 Kesimpulan

Program python dapat berjalan ataupun tidak berjalan berkat adanya lexer dan parser. Lexer menerima input berupa karakter sedangkan parser menerima input dalam bentuk token dari lexer. Kedua program ini bekerja sama untuk mengecek dan memastikan bahwa kode program yang dibuat oleh programmer mengikuti aturan dari bahasa Python. Kedua program inilah yang mampu menampilkan pesan syntax error jika terjadi kesalahan syntax pada saat program di run. Konsep kerja dari keduanya adalah finite automata, regex dan CFG. Algoritma yang bekerja pada masing-masing parser dapat berbeda akan tetapi kami menggunakan algoritma CYK yang menerima grammar CNF. Kami juga tidak secara langsung membuat grammar dalam bentuk CNF melainkan dalam bentuk CFG dan membuat grammar converter untuk memudahkan pengerjaan. Program kami dapat mengecek input file dan mengirimkan output berupa accepted jika tidak ada syntax error, dan output berupa jumlah error beserta kesalahan syntaxnya jika ada error. Tugas besar ini mengajarkan kami banyak sekali pengetahuan akan cara kerja parser dan lexer serta meningkatkan rasa syukur atas adanya kedua program tersebut. Peran parser dan lexer sangatlah besar dalam dunia pemrograman karena keduanya memudahkan programmer dalam mencari letak dan jenis kesalahan pada program.

4.2 Saran

Pengerjaan tugas besar ini tentu tidak luput dari kesalahan. Salah satu kekeliruan kami ialah tidak membuat DFA dari awal sehingga pengecekan syntax variabel menjadi lebih susah. Riset akan parser dan lexer sebaiknya dilakukan lebih mendalam agar lebih mengetahui cara kerja keduanya secara lebih baik.

Lampiran

Repository Github

https://github.com/NayotamaPradipta/AgenParserTBFO

Pembagian Tugas

Nayotama Pradipta - 13520089	Membuat file grammar converter, lexerRules, modifikasi parser, membuat dan mengisi main, testing program, pengerjaan laporan
Angelica Winasta Sinisuka - 13520097	testing program, pengerjaan laporan, pembuatan FA, membantu pengerjaan main
Muhammad Rakha Athaya - 13520108	testing program, membuat file grammar, pengerjaan laporan

Daftar Referensi

1. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory , Languages, and Computation, Third Edition, Addison Wesley, 2014
2. https://github.com/RobMcH/CYK-Parser
3. https://docs.python.org/3/reference/lexical_analysis.html
4. https://gist.github.com/eliben/5797351
5. https://www.pythonindo.com/source-code-deterministic-finite-automaton-dfa-menggunakan-python/