

Laporan Tugas Besar
IF2124 Teori Bahasa Formal dan Otomata
Compiler Bahasa Python
Semester I Tahun 2021/2022

Disusun oleh:

Jevant Jedidia Augustine	13520133
Gerald Abraham Sianturi	13520138
Atabik Muhammad Azfa Shofi	13520159



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

Daftar Isi

Daftar Isi	1
Bab 1 Teori Dasar	2
1.1 Finite Automata (FA)	2
1.2 Context Free Grammar (CFG)	2
1.3 Syntax Python	2
Variable	2
If, Elif, dan Else	3
While	3
For	3
From dan Import	3
With	3
Def	3
Class	4
Keyword Lainnya	4
Bab 2 Hasil CFG dan FA	5
2.1 CFG	5
2.2 FA	6
Bab 3 Implementasi	7
Bab 4 Sampel Pengujian	9
4.1 Kasus Uji 1	9
4.2 Kasus Uji 2	9
4.3 Kasus Uji 3	10
4.4 Kasus Uji 4	11
Known Bugs	12
Referensi	13
Lampiran	14

Bab 1 Teori Dasar

1.1 Finite Automata (FA)

Finite Automata (FA) adalah sebuah mesin sederhana yang dibuat untuk mengenal suatu pola. Finite automata menerima atau menolak input berdasarkan pola yang diterima oleh FA. Suatu FA terdiri atas tuple yang berisikan 5 elemen, yaitu sebagai berikut:

$$FA = \{Q, \Sigma, q, F, \delta\}$$

Q = himpunan *state* yang berhingga

Σ = himpunan simbol input

q = *state* awal

F = himpunan *state* akhir

δ = fungsi transisi

FA dapat dibagi menjadi 2 jenis, yaitu Deterministic Finite Automata (DFA) dan Nondeterministic Finite Automata (NFA). Pada DFA, untuk input suatu karakter, mesin akan pergi ke satu *state* saja. Fungsi transisi terdefinisi untuk setiap *state* untuk input suatu simbol. DFA juga tidak menerima input kosong (ϵ). NFA mirip dengan DFA kecuali untuk beberapa hal, yaitu input kosong pada NFA diperbolehkan dan bila mesin menerima suatu input, mesin dapat pergi ke beberapa *state*.

1.2 Context Free Grammar (CFG)

Context Free Grammar (CFG) didefinisikan dengan tuple yang berisikan 4 elemen, yaitu:

$$G = \{N, T, P, S\}$$

N = himpunan simbol *non-terminal*

T = himpunan simbol *terminal*

P = aturan produksi

S = simbol awal

Dalam CFG, simbol awal digunakan untuk menurunkan input string. String dapat diturunkan berulang kali dengan mengganti simbol *non-terminal* sesuai dengan aturan produksi. Penurunan dilakukan hingga seluruh input string menjadi simbol *terminal*.

1.3 Syntax Python

Variable

Penamaan variable di Python memiliki aturan sebagai berikut:

- Harus diawali dengan huruf atau *underscore*

- Tidak boleh diawali dengan angka
- Hanya terdiri dari karakter *alpha-numeric* dan *underscore*

If, Elif, dan Else

Syntax dari *if*, *elif*, dan *else* adalah:

```
if condition:
    Body of if
elif condition:
    Body of elif
else:
    Body of else
```

While

Syntax dari *while* adalah:

```
while condition:
    Body of while
```

For

Syntax dari *for* adalah:

```
for val in sequence:
    Body of for
```

From dan Import

Syntax dari *from* dan *import* adalah:

```
from variable import variable as variable
```

With

Syntax dari *with* adalah:

```
with function as variable:
    Body of with
```

Def

Syntax dari *def* adalah:

```
def variable (arguments of function):
    Body of def
```

Class

Syntax dari class adalah:

```
class variable (arguments of class):  
    Body of class
```

Keyword Lainnya

- `False` dan `True` merupakan boolean yang merupakan condition
- `not`, `and`, dan `or` merupakan operasi *logic* yang dapat membentuk suatu condition/boolean
- `break` dan `continue` dapat digunakan hanya pada loop, yaitu `while` dan `for`
- `return` digunakan pada body of `def`

Bab 2 Hasil CFG dan FA

2.1 CFG

```
START → IF SEC | DEF SEC | IMPORT SEC | WITH SEC | CLASS SEC | WHILE SEC
SEC → IF SEC | DEF SEC | IMPORT SEC | WITH SEC | CLASS SEC | WHILE SEC | ε
STATEMENT → PASS STATEMENTSEC | FUNCTIONCALL STATEMENTSEC | ASSIGNMENT STATEMENTSEC | IF STATEMENTSEC | WHILE STATEMENTSEC | FOR STATEMENTSEC
STATEMENTSEC → PASS STATEMENTSEC | FUNCTIONCALL STATEMENTSEC | ASSIGNMENT STATEMENTSEC | IF STATEMENTSEC | WHILE STATEMENTSEC | FOR STATEMENTSEC | ε
CONTENTLOOP → STATEMENT NULLCONTENTLOOP | BREAK NULLCONTENTLOOP | CONTINUE NULLCONTENTLOOP
NULLCONTENTLOOP → STATEMENT NULLCONTENTLOOP | BREAK NULLCONTENTLOOP | CONTINUE NULLCONTENTLOOP | ε
FUNCTIONCALL → VARIABLE ( ARGUMENTSOFFUNCCALL ) | VARIABLE DOT_LOOP ( ARGUMENTSOFFUNCCALL )
DOT_LOOP → . VARIABLE DOT_LOOP | ε
ARGUMENTSOFFUNCCALL → DATA LOOPFUNC | ε
LOOPFUNC → , DATA LOOPFUNC | ε
ARGUMENTSOFFUNCDEF → VARIABLE LOOPFUNCDEF | ε
LOOPFUNCDEF → , VARIABLE LOOPFUNCDEF | ε
DATA → INT | VARIABLE | BOOLEAN | FUNCTIONCALL
DATA_X → INT | VARIABLE | BOOLEAN | FUNCTIONCALL | NONE | “ STRING ”
ASSIGNMENT → VARIABLE ASSIGN VALUE
RETURN → return DATA | ε
OPERATOR → BIT | ARITHMETIC | LOGIC
LOGIC → == | <= | >= | != | < | > | <>
ARITHMETIC → + | - | * | / | ** | // | %
ASSIGN → = | += | -= | *= | /= | **= | //= | %= | &= | |= | ^= | >>= | <<=
BIT → & | | | ^ | ~ | << | >>
```

```
VALUE → DATA OPERATOR VALUE | DATA OPERATOR DATA | DATA
# False & True & not & is & and & or
BOOLEANBASE → True | False | VARIABLE | FUNCTIONCALL | BOOLOPERATOR | IS | ORAND
BOOLOPERATOR → NOT ( DATA ) | LOGIC NOT ( DATA ) | NOT DATA | LOGIC NOT ( DATA ) | NOT ( DATA ) | LOGIC NOT DATA | NOT DATA_X | LOGIC NOT DATA_X
BOOLEAN → NOT ( BOOLEAN ) | NOT BOOLEAN | NOT ( BOOLEANBASE ) | NOT BOOLEANBASE
NOT → not | ε
# Is
IS → VARIABLE is NOT VARIABLE | VARIABLE is ( NOT VARIABLE )
ORAND → BOOLEAN ORAND BOOLEAN
ORANDT → or | and
# None
NONE → None
# Class
CLASS → class VARIABLE ( ARGUMENTSOFFUNC ) : SECCCLASS
SECCCLASS → DEF SUBSECCCLASS | STATEMENT SUBSECCCLASS
SUBSECCCLASS → DEF SUBSECCCLASS | STATEMENT SUBSECCCLASS | ε
# With
WITH → with FUNCTION as VARIABLE : STATEMENT
# Def
DEF → def VARIABLE ( ARGUMENTSOFFUNCDEF ) : STATEMENT RETURN
# for
FOR → for VARIABLE in SEQUENCE : CONTENTLOOP
SEQUENCE → VARIABLE | “ STRING ” | range ( INT ) | range ( INT , INT ) | range ( INT , INT , INT )
# While
WHILE → while ( BOOLEAN ) : CONTENTLOOP | while BOOLEAN : CONTENTLOOP
If & Elif & Else
IF → if BOOLEAN : STATEMENT BRANCH | if ( BOOLEAN ) : STATEMENT BRANCH
BRANCH → elif BOOLEAN : STATEMENT BRANCH | elif ( BOOLEAN ) : STATEMENT BRANCH | else BOOLEAN : STATEMENT | else ( BOOLEAN ) : STATEMENT | ε
From & Import & As
IMPORT → FROM import VARIABLE AS
AS → as VARIABLE | ε
FROM → from VARIABLE | ε
```

```

# Break
BREAK → break BREAK | ε
# Continue
CONTINUE → continue CONTINUE | ε
# Pass
PASS → pass PASS | ε

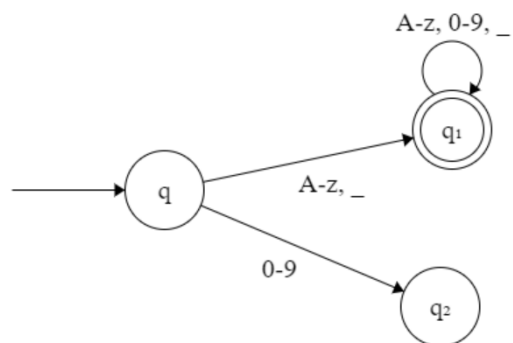
```

2.2 FA

```

Syntax Variable
FA = ({q, q1, q2}, {A-z, 0-9, _}, q, q1, δ)
1. δ(q, 0-9) = (q2)
2. δ(q, A-z) = (q1)
3. δ(q, _) = (q1)
4. δ(q1, A-z) = (q1)
5. δ(q1, 0-9) = (q1)
6. δ(q1, _) = (q1)

```



Bab 3 Implementasi

- parserprogram.py

File main adalah file untuk menjalankan parser terhadap file yang dipilih dengan menggunakan fungsi-fungsi yang telah dibuat pada file cyk dan process_file.

- cyk.py

File ini berisikan CFG yang telah dibuat, dalam bentuk CNF berisikan Rules grammar. Terdapat satu fungsi dalam file ini yaitu cykParse yang menerima array lalu mengeluarkan apakah array tersebut terdapat dalam grammar atau tidak.

- process_file.py

File ini adalah yang paling penting karena berisikan fungsi-fungsi yang berguna memproses file menjadi array sehingga siap digunakan sebagai argumen fungsi cyk. Fungsi-fungsi tersebut antara lain:


- cekAngka : menghasilkan true jika input adalah angka
- cekLower : menghasilkan true jika input adalah huruf a-z
- cekUpper : menghasilkan true jika input adalah huruf A-Z
- cekVariabel : menghasilkan true jika input memenuhi syarat variabel di Python
- cekArrVar : menghasilkan true jika semua anggota array adalah variabel
- splitColon : melakukan split terhadap titik dua (":")
- splitKurungBuka : melakukan split terhadap kurung buka ("(")
- splitKurungTutup : melakukan split terhadap kurung tutup (")")
- splitKoma : melakukan split terhadap koma (",")
- splitTitik : melakukan split terhadap titik (".")
- split : melakukan split dengan memanggil fungsi split yang lain
- cutMid : menghapus array pada indeks i ke indeks j dimana $j > i$
- delComment : menghapus multiline comment pada file input

- getNonTerm : mendapatkan nilai yang tidak terdapat pada terminal CNF
- getInt : mendapatkan nilai bertipe integer pada input array
- getFloat : mendapatkan nilai bertipe float pada input array
- getString : mendapatkan nilai bertipe string pada input array
- getVariable : mendapatkan nilai selain int, float dan string pada input array
- getData : mendapatkan data variabel, string dan angka (int dan float)
- Replace : mengganti nilai bertipe string dengan “word”, bertipe angka dengan “num”, variabel dengan “var”

Bab 4 Sampel Pengujian

4.1 Kasus Uji 1

```
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"
```




```
>> python parserprogram.py  
Masukkan nama file: test.py  
Accepted
```

Kode di atas diterima oleh compiler karena kode sudah sesuai dengan *syntax python*.

4.2 Kasus Uji 2

```
def do_something(x):  
    x + 2 = 3  
    ''' 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"
```




```
>> python parserprogram.py
Masukkan nama file: test.py
Accepted
```

Kode di atas tidak diterima oleh *compiler* karena memiliki *syntax error*. *Syntax error* terdapat pada line “ $x + 2 = 3$ ”, operasi aritmatik hanya dapat dituliskan di bagian kiri assignment.

4.3 Kasus Uji 3



```
for i in range(4):
    square(4)
```



```
>> python parserprogram.py
Masukkan nama file: test.py
['i', 'square']
['for', 'var', 'in', 'range', '(', 'num', ')', ':', 'var', '(', 'num', ')']
Accepted
```

Kode di atas diterima oleh *compiler*; meskipun fungsi `square` belum didefinisikan. Hal ini terjadi karena tuntutan program hanya memperhatikan *syntax* dan mengabaikan arti semantik.

```
for i in range(4):  
    print("test")
```

```
>> python parserprogram.py  
Masukkan nama file: test.py  
['i', 'print']  
['for', 'var', 'in', 'range', '(', 'num', ')', ':', 'var', '(', 'word', ')']  
Accepted
```

4.4 Kasus Uji 4

```
while i < 3:  
    print(xSquare)
```

```
>> python parserprogram.py  
Masukkan nama file: test.py  
Accepted
```

Kode di atas diterima oleh *compiler*, meskipun variabel *i* dan *xSquare* belum didefinisikan. Hal ini terjadi karena tuntutan program hanya memperhatikan *syntax* dan mengabaikan arti semantik.

Known Bugs


1. Tipe data dasar, seperti `int`, `string`, dsb. dapat di-*handle* pada program ini, tetapi struktur data yang lebih kompleks, seperti array, set, *abstract data type*, dsb. deklarasinya tidak dapat di-*handle*

Contoh:



```
a = [1, 2, 3, 4, 5]
```

Akan menghasilkan:



```
>> python parserprogram.py
Masukkan nama file: test.py
['a', '[1', '5']']
Syntax error
```

Referensi

- [1] <https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/>
- [2] <https://realpython.com/python-data-types/>

Lampiran

- [Link repository](#)
- [Pembagian kerja](#)