TUGAS BESAR

IF2124 TEORI BAHASA FORMAL DAN OTOMATA

PARSER BAHASA JAVASCRIPT (NODE.JS)

SEMESTER 1 TAHUN 2022/2023

Disusun Oleh

Kelompok 1 AverageTBFOEnjoyers

13521110 Yanuar Sano Nur Rasyid

13521112 Rayhan Hanif Maulana Pradana

13521173 Dewana Gustavus Haraka Otang

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

BAB I

Teori Dasar

1.1 Finite Automata

Finite automata merupakan sebuah mesin abstrak untuk mengenali bahasa paling sederhana atau regular language. Mesin ini dapat menerima input , mengubah suatu keadaan atau state menjadi state yang lain, dan mengeluarkan output. Finite automata atau FA terdiri atas lima elemen tuple yaitu

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

Q = himpunan state

Σ = himpunan simbol input

δ = fungsi transisi $\delta : Q \times \Sigma$

S = state awal / initial state , $S \in Q$

F = state akhir / final state, $F \subseteq Q$

FA akan menerima sebuah string simbol input yang akan mengubah suatu state pada automata sesuai dengan fungsi transisi hingga semua input terbaca. Penentuan string yang dimasukkan diterima atau tidak ditentukan oleh state akhir automata. Jika merupakan final state maka input diterima dan sebaliknya jika bukan merupakan final state maka input ditolak.

FA terbagi menjadi dua jenis yaitu

1. Deterministic Finite Automata (DFA)

   Merupakan sebuah FA yang hanya dapat menuju kepada satu state dari input yang diterima. Selain itu, DFA juga tidak menerima input kosong atau Null.

2. Nondeterministic Finite Automata (NFA)

   Merupakan sebuah FA yang dapat menerima input Null atau kosong dan dapat menuju ke lebih dari satu state saat menerima suatu input.

1.2 Context Free Grammar

Context Free Grammar atau CFG adalah sebuah tata bahasa formal yang dapat menunjukkan semua kemungkinan string yang dapat dibentuk dari sebuah bahasa formal.

CFG dapat didefinisikan menjadi empat buah tupel yaitu

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

T = himpunan simbol terminal

V = himpunan simbol non-terminal

P = aturan production

S = start simbol

CFG memiliki produksi berbentuk A -> B dengan A adalah sebuah variabel dan B dapat berupa variabel , terminal , atau Null. Berikut adalah contoh dari produksi CFG

S -> A B C | B

A -> 'a' C | B | ε

B -> C | 'b' S

C -> A | ε

Salah satu algoritma parsing CFG adalah algoritma CYK atau Cocke–Younger–Kasami. CYK merupakan algoritma yang memanfaatkan dynamic programming untuk menafsirkan suatu CFG dalam bentuk Chomsky Normal Form atau CNF.

1.3 Chomksy Normal Form

CNF merupakan sebuah CFG yang aturan produksinya memenuhi syarat berikut.

1. Start symbol menghasilkan ε.
2. Non-terminal symbol menghasilkan 2 non-terminal.
3. Non-terminal menghasilkan satu simbol terminal.

Berikut adalah contoh dari CNF.

S -> A B | B C

A -> B A | 'a'

B -> C C | 'b'

C -> A B | 'a'

Untuk mengkonversi sebuah CFG menjadi CNF dapat dilakukan langkah-langkah berikut.

1. Eliminasi Null Production
2. Eliminasi Unit Production
3. Eliminasi Useless Production
4. Membuat Produksi menjadi dua variabel atau satu terminal

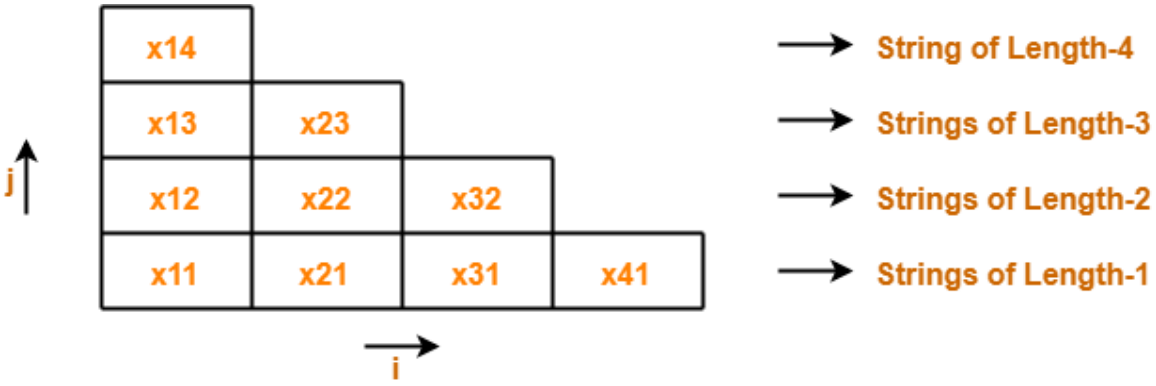
1.4 Algoritma CYK

Algoritma CYK digunakan untuk menentukan apakah sebuah string input dapat diterima oleh grammar yang diberikan, algoritma CYK membuat tabel berbentuk segitiga dengan notasi seperti pada gambar, notasi x_{ij} berarti substring yang dimulai pada index I dan memiliki panjang j, nilai pada tabel berisi kumpulan variabel yang dapat meraih substring x_{ij} , cara kerja algoritma CYK adalah sebagai berikut :

1. Isi tabel paling bawah dengan variabel yang bisa menurunkan alphabet pada x_{i1}
2. Untuk setiap substring x_{ij} selain x_{i1} lakukan cartesian product $x_{ik} * x_{(i+k)(j-k)}$ untuk setiap $0 < k < j$
3. Ubah variabel yang didapat dari hasil cartesian product menjadi variabel yang bisa menurunkan variabel tersebut (jika ada)
4. Ulangi langkah 2 dan 3 sampai seluruh tabel sudah terisi variabel

5. Apabila pada kotak teratas (x_{1n} dengan n adalah panjang string) terdapat variabel S, maka string diterima oleh grammar

Berikut adalah contoh tabel dynamic programming saat menjalankan algoritma CYK.



Gambar 1.4.1 Tabel Algoritma CYK

Time complexity yang didapat ketika memakai algoritma CYK adalah $O(n^3 * |G|)$ dengan n adalah panjang string input dan |G| adalah jumlah grammar pada CNF.

1.5 Syntax Javascript

Javascript (JS) adalah sebuah bahasa pemrograman "Turing Complete" dan "dynamically typed" dengan paradigma Berorientasi Objek, Fungsional, dan Imperatif. JS umumnya digunakan dalam pengembangan web, dimana sekitar 98% dari website menggunakan JS untuk perilaku webpage pada client-side.

Syntax-syntax dasar dari JS berupa :

1. Deklarasi

Deklarasi variabel pada JS diawali dengan 'let', 'var', atau 'const' sebelum variabel, seperti berikut

```
let x = 1;
```

Gambar 1.5.1.2 Variabel let

```
var x = 1;
```

Gambar 1.5.1.2 Variabel var

```
const number = 42;
```

Gambar 1.5.1.3 Variabel const

2. Object

Object pada sebuah statement dapat dipanggil dengan atribut/properti ataupun method dalam objek tersebut

```
objectName.propertyName
```

Gambar 1.5.2.1 Property object

```
objectName.methodName()
```

Gambar 1.5.2.2 Method object

Dalam method pada object juga dapat diberi argumen/parameter

3. Function

Fungsi pada JS didefinisikan/deklarasikan dengan ‘function’, kemudian dilanjutkan dengan nama fungsi dan parameter/argumen. Fungsi kemudian akan memiliki sebuah ‘body’ yang diawali dan diakhiri kurawal dan berisi kode program, seperti berikut

```
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Gambar 1.5.3 Fungsi javascript

Dapat dilihat bahwa fungsi juga dapat mengembalikan (return) sebuah nilai.

4. Block

Block pada javascript merupakan sebuah blok dari kode yang diawali dan diakhiri dengan kurung kurawal

```
{
  foo(); // Logs "foo"
  function foo() {
    console.log("foo");
  }
}
```

Gambar 1.5.4 Block javascript

5. Control Flow

a) If..else

if..else (conditional branching) seperti pada bahasa pemrograman umumnya digunakan untuk menjalankan sebuah blok kode program apabila memenuhi sebuah kondisi

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Gambar 1.5.5.1 If else javascript

b) Switch

Switch merupakan alternatif dari if..else

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Gambar 1.5.5.2 Switch javascript

c) Break dan Continue

Break dan Continue digunakan dalam looping/iterasi, dimana break digunakan untuk menghentikan iterasi jika memenuhi suatu kondisi dan continue digunakan untuk melanjutkan iterasi.

Break : break;

Continue : continue;

d) Error handling

Error handling berfungsi untuk mencegah keluaran error/exception pada stdout dan menggantikannya dengan keluaran atau perintah yang dibuat oleh pemrogram.

Throw : Merupakan exception yang didefinisikan sendiri oleh pemrogram

```
throw "Too big";
throw 500;
```

Gambar 1.5.5.3 Throw javascript

Try : Mengetes sebuah blok kode program

Catch : Blok kode program yang dijalankan jika program pada try mengalami error

Finally : Blok kode program yang tetap dijalankan meskipun blok program error ataupun tidak

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

Gambar 1.5.5.4 Throw javascript

6. Iterasi/Loop

Dua jenis dasar iterasi adalah for loop dan while loop seperti berikut

```
for (expression 1; expression 2; expression 3) {
  // code block to be executed
}
```

Gambar 1.5.6.1 For loop javascript

```
while (condition) {
  // code block to be executed
}
```

Gambar 1.5.6.1 While loop javascript

7. Operator

a) Aritmatika

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Gambar 1.5.7.1 Operator Aritmatika javascript

b) Logical

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Gambar 1.5.7.2 Operator Logical javascript

c) Comparison

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Gambar 1.5.7.3 Operator Comparison javascript

d) Bitwise

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Gambar 1.5.7.4 Operator Bitwise javascript

BAB II

FA dan CFG

2.1 NFA Variabel

FA untuk memeriksa apakah string memenuhi syarat penamaan variabel di *javascript* atau tidak diimplementasikan dalam sebuah NFA yang memiliki komponen seperti berikut.

2.1.1 Q atau himpunan state-nya

```
start_state_var = 0
final_state_var = 1
```

Gambar 2.1.1 Himpunan state NFA variabel

S = state awal / initial state , S ∈ Q = 0

F = state akhir / final state, F ⊆ Q = 1

State 0 juga menunjukkan bahwa mesin belum menerima input sementara state 1 menunjukkan bahwa mesin sudah menerima input yang valid.

2.1.2 Σ atau himpunan simbol input

```
lowercase = string.ascii_lowercase
uppercase = string.ascii_uppercase
all_letter = lowercase + uppercase
digits = string.digits

other_char = ['_', '$']

first_char = list(all_letter) + other_char
second_char = first_char + list(digits)


reserved_name = ['break', 'default', 'for', 'return' , 'var' , 'const' , 'delete',
                 'function', 'switch', 'while', 'case', 'else', 'if', 'throw', 'catch',
                 'false', 'let', 'try', 'continue' , 'finally' , 'null', 'true']
```

Gambar 2.1.2 Himpunan simbol input NFA variabel

First_char yang terdiri atas huruf kecil , huruf besar , dan ‘_’ , ‘$’.

Second_char yang terdiri atas first_char dan digit.

Kemudian ada himpunan reserved_name yang berisi nama yang tidak bisa dipakai sebagai sebuah variabel.

2.1.3 δ atau fungsi transisi δ : Q × Σ

```
transition_variable = {}

# start state
transition_variable[start_state_var] = {}
for char in first_char:
    transition_variable[start_state_var][char] = 1

# final state
transition_variable[final_state_var] = {}
for char in second_char:
    transition_variable[final_state_var][char] = 1
```

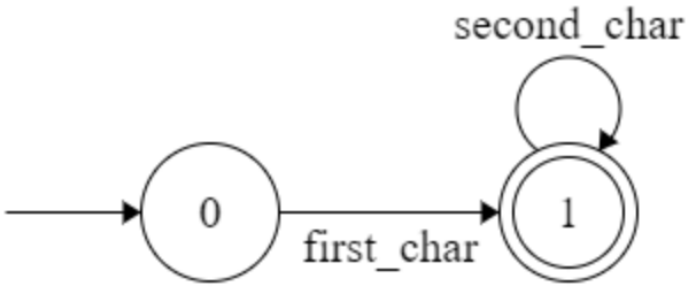
Gambar 2.1.3 Fungsi transisi NFA variabel

Fungsi transisi dimplementasikan dalam sebuah dictionary of dictionary.

δ (0, first_char) = {1}

δ (1, second_char) = {1}

2.1.4 Diagram NFA



Gambar 2.1.4 Diagram NFA variabel

2.2 NFA Operasi

FA untuk memeriksa apakah string operasi memenuhi syarat-syarat operasi di *javascript* atau tidak diimplementasikan dalam sebuah NFA yang memiliki komponen seperti berikut.

2.2.1 Q atau himpunan state

```
start_state_op = 0
final_state_op = 1
secondary_state_op = 2
ternary_state1_op = 3
```

Gambar 2.2.1 Himpunan state NFA operasi

S = state awal / initial state , S ∈ Q = 0

F = state akhir / final state, F ⊆ Q = 1

State 0 juga menunjukkan bahwa mesin belum menerima input

State 1 menunjukkan bahwa mesin sudah menerima input yang valid yaitu sebuah angka atau variabel.

State 2 menujukkan bahwa mesin sudah menerima input operator.

State 3 menunjukkan bahwa mesin sudah menerimua input ‘:’ dari ternery operator

2.2.2 Σ atau himpunan simbol input

```
operator = ['+', '-', '*', '/', '%', '+=',
            '-=', '*=', '/=', '%=', '==', '!=', '===',
            '!==', '>', '<', '>=', '<=', '&&', '||',
            '&', '|', '^', '<<', '>>', '>>>',
            '<<=', '>>=', '>>>=', '=>', '=']

unary_operator = ['+', '-', '++', '--', '!', '~']

ternary_op = ['?', ':']
```

Gambar 2.2.2 Himpunan input simbol NFA operasi

Operator yang mencakup semua operasi di javacript.

Unary_operator yang mencakup operasi satu angka atau variabel di javacript

Ternary_op yang mencakup ternary operation di javacript.

Juga memakai input simbol second_char dari NFA variabel.

2.2.3 δ atau fungsi transisi δ : Q × Σ

```
# start state
transition_operator[start_state_op] = {}
for char in second_char:
    transition_operator[start_state_op][char] = 1

# final state
transition_operator[final_state_op] = {}
for char in operator:
    transition_operator[final_state_op][char] = 2
for char in unary_operator:
    transition_operator[final_state_op][char] = 1
for char in second_char:
    transition_operator[final_state_op][char] = 1
transition_operator[final_state_op][ternary_op[0]] = 3

# secondary state
transition_operator[secondary_state_op] = {}
for char in second_char:
    transition_operator[secondary_state_op][char] = 1

# ternary state1
transition_operator[ternary_state1_op] = {}
for char in second_char:
    transition_operator[ternary_state1_op][char] = 3
transition_operator[ternary_state1_op][ternary_op[1]] = 2
```

Gambar 2.2.3 Fungsi transisi NFA operasi

δ (0, second_char) = {1}

δ (1, second_char) = {1}

δ (1, unary_operator) = {1}

δ (1, operator) = {2}

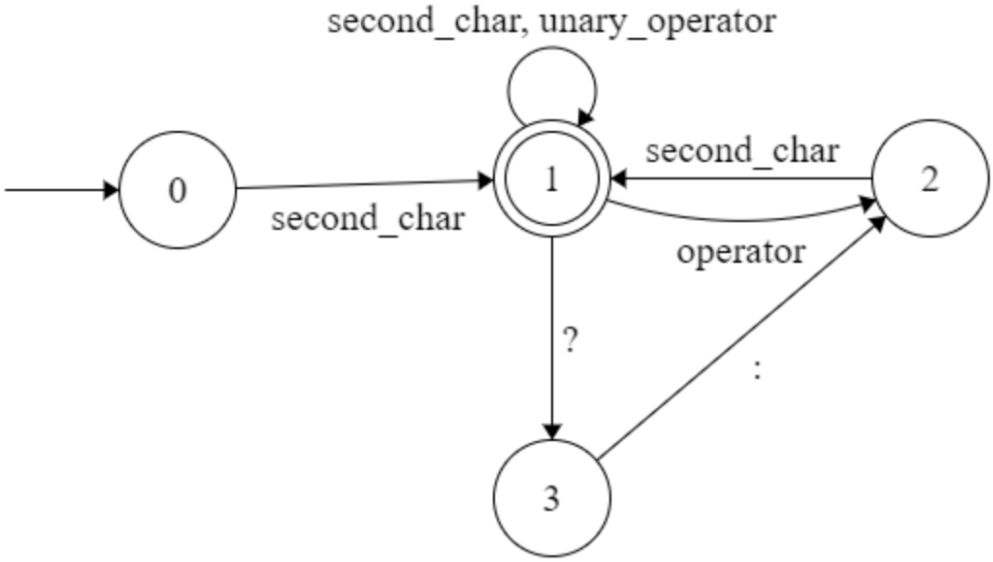
δ (1, ‘?’) = {3}

δ (2, second_char) = {1}

δ (3, second_char) = {3}

δ (3,’:’) = {2}

2.2.4 Diagram NFA



Gambar 2.2.4 Diagram NFA operasi

2.3 CFG Syntax Javascript

2.3.1 Variabel

Variabel yang diguakan pada CFG parsing JS berupa huruf alfabet kapital (dengan angka dalam beberapa kasus), seperti berikut

```
PARAM
ASSIGN3
```

2.3.2 Terminal

Simbol terminal yang digunakan pada CFG parsing JS berupa sebuah karakter ASCII yang ditutup dengan tanda kutip.

2.3.3 Start Simbol

```
S -> STATEMENT1 S | BODY1 S | FUNCTION S | SPACEOREMPTY
```

Start simbol S akan menurunkan tiga variabel utama, yaitu STATEMENT1, BODY1, FUNCTION, lalu menurunkan start simbol itu sendiri. S juga dapat menurunkan SPACEOREMPTY sendiri.

2.3.4 STATEMENT1

Variabel STATEMENT1 menurunkan statement-statement dasar pada JS, yaitu deklarasi, dan error handling. STATEMENT1 juga dapat menurunkan SPACEOREMPTY sendiri. STATEMENT1 tidak akan menurunkan return.

2.3.5 BODY1

Variabel BODY1 akan menurunkan sebuah block, yaitu STATEMENT1 yang ditutup dengan kurung kurawal.

```
BODY1 -> '{' SPACEOREMPTY STATEMENT1 SPACEOREMPTY '}'
```

2.3.6 FUNCTION

Variabel FUNCTION menurunkan deklarasi/definisi fungsi. Setelah deklarasi fungsi, FUNCTION akan menurunkan variabel FUNCBODY yang pada dasarnya akan menurunkan statement-statement JS dan return (return tidak diturunkan oleh variabel STATEMENT1).

2.3.7 SPACEOREMPTY

Variabel SPACEOREMPTY pada dasarnya bertujuan untuk menurunkan spasi (whitespace) ataupun simbol kosong (epsilon). Berikut adalah penurunan SPACEOREMPTY.

```
SPACE -> ' ' | ' ' SPACE
SPACEOREMPTY -> SPACE | epsilon
```

Dapat dilihat bahwa terdapat variabel SPACE untuk menurunkan lebih dari satu spasi.

BAB III

Implementasi dan Pengujian

3.1 Implementasi fa.py

Struktur data yang digunakan pada implementasi ini adalah:

1. List
   List digunakan sebagai tempat menyimpan himpunan simbol input , seperti huruf, digit, dan karakter lain. List ini juga berguna untuk menyimpan daftar nama variabel yang sudah dipakai oleh *javascript*.
2. Dictionary
   Dictionary digunakan sebagai tempat menyimpan hubungan relasi antara input simbol dan state sehingga berfungsi sebagai fungsi transisi pada Finite Automata

```
def is_reserved_name(variable):
```

Fungsi is_reserved_name menerima input sebuah string yang merupakan sebuah nama variabel dan akan mengirim boolean yang akan bernilai true apabila string yang diberikan adalah keyword javascript yang valid (contoh : break, for, if, dll).

```
def is_legal_variable(variable):
```

Fungsi is_legal_variable menerima input sebuah string yang merupakan sebuah nama variabel dan akan mengirim boolean yang akan bernilai true apabila setiap character pada string dapat diterima pada FA untuk mengecek nama sebuah variabel, string yang diberikan juga tidak boleh ada di dalam reversed name (keyword javascript).

```
def split_operation(operation):
```

Fungsi split_operation menerima input sebuah string yang merupakan kumpulan operasi aritmatika dan akan mengirimkan sebuah list yang berisi kumpulan string yang merupakan string input yang sudah dipisah berdasarkan variabel dan operator.

```
def is_legal_operation(operation):
```

Fungsi is_legal_operation menerima input sebuah string yang merupakan kumpulan operasi aritmatika dan akan mengirimkan sebuah boolean yang akan bernilai true apabila input yang diberikan merupakan operasi aritmatika yang valid.

3.2 Implementasi cnf_convert.py

Struktur Data

1. List
   List digunakan sebagai tempat menyimpan himpunan variabel dan juga terminal dari CFG yang sudah dibaca.
2. Set
   Set digunakan sebagai tempat penyimpanan variabel yang memiliki produksi epsilon dan tidak terdapat duplikat.
3. Dictionary

Dictionary digunakan sebagai tempat penyimpanan produksi dengan key sebagai variabel awal dan list of value sebagai semua hasil produksi dari variabel tersebut.

```
def parsingCFG(file):
```

Fungsi parsingCFG menerima input path sebuah file dan akan membaca sebuah file berisi CFG dan memisahkan simbol-simbol pada grammar kedalam sebuah dictionary yang nilai pada keynya merupakan variabel produktor pada aturan produksi dan valuenya adalah hasil produksi.

```
def productList(word, from_char, to_char):
```

Fungsi productList menerima input sebuah string dan 2 buah char yang merupakan char awal dan char hasil, fungsi akan menghasilkan seluruh kemungkinan string apabila kemunculan char awal pada string terdapat 2 kemungkinan yaitu char awal dibiarkan atau char awal diganti dengan char akhir.

```
def removeNull(grammar):
```

Fungsi removeNull menerima input sebuah grammar CFG dan akan menghasilkan grammar CFG yang tidak menghasilkan null, dengan cara menandai seluruh produksi yang dapat menghasilkan null dan juga dapat menghasilkan produksi yang dapat menghasilkan null, serta untuk setiap kemunculan variabel yang dapat menghasilkan null akan diubah menjadi seluruh kemungkinan dimana variabel itu dapat tetap dijadikan hasil produksi atau diabaikan.

```
def removeDuplicateProd(grammar):
```

Fungsi removeDuplicateProd menerima input sebuah grammar CFG dan menghasilkan grammar yang hasil produksinya tidak mengandung duplikat.

```
def removeNoProd(grammar):
```

Fungsi removeNoProd menerima input sebuah grammar CFG dan menghasilkan grammar dimana produksi yang tidak memiliki hasil produksi dihapus dari grammar.

```
def removeBlankProd(grammar):
```

Fungsi removeBlankProd menerima input sebuah grammar CFG dan menghasilkan grammar dimana produksi yang memiliki hasil produksi string kosong dihapus dari grammar.

```
def replaceEpsilon(grammar):
```

Fungsi replaceEpsilon menerima input sebuah grammar CFG dan menghasilkan grammar CFG dimana hasil produksi epsilon dihapus dari grammar.

```
def removeUnit(grammar):
```

Fungsi removeUnit menerima input sebuah grammar CFG dan menghasilkan grammar CFG yang seluruh hasil produksi yang hanya mengandung 1 variabel produksi (unit production) dihilangkan dan akan ditambah seluruh hasil produksi unit production ke dalam hasil produksi.

```
def replaceTerminal(grammar):
```

Fungsi replaceTerminal menerima input sebuah grammar CFG dan menghasilkan grammar CFG yang seluruh hasil produksi yang mengandung lebih dari 1 terminal akan diubah dengan cara terminal dijadikan variabel dan membuat variabel baru yang dapat menghasilkan hanya 1 terminal yang diubah.

```
def isTerminal(prod):
```

Fungsi isTerminal menerima input sebuah hasil produksi dan akan mengirim boolean yang akan bernilai true apabila hasil produksi adalah terminal yaitu yang diawali dengan kutip, terdapat sebuah character ditengah, dan diakhiri dengan kutip.

```
def removeInvalid(grammar):
```

Fungsi removeInvalid menerima input sebuah grammar CFG dan menghasilkan grammar CFG yang sudah tidak memiliki produk invalid untuk sebuah CNF yaitu tidak memiliki produksi hanya suatu variabel atau produksi terminal lebih dari satu.

```
def makeTwoVar(grammar):
```

Fungsi makeTwoVar menerima input sebuah grammar CFG dan akan menghasilkan grammar CFG yang seluruh hasil produksi yang mengandung lebih dari 2 variabel diubah menjadi hanya 2 variabel dan produksi baru yang menjembatani produksi semula agar bisa dijadikan hanya 2 variabel.

```
def printgrammar(grammar):
```

Prosedur printgrammar menerima input sebuah grammar CFG dan akan menampilkan grammar

3.3 Implementasi CYK_Algorithm.py

Struktur Data

1. List
   List digunakan untuk menyimpan variabel dan terminal apa saja yang dapat diproduksi pada grammar, dan juga digunakan untuk membuat tabel CYK yang merupakan matrix of set
2. Set
   Set digunakan untuk menyimpan variabel yang dapat menurunkan sebuah substring pada tabel CYK, pemakaian set dilakukan agar mencegah adanya duplikat sehingga perhitungan akan lebih cepat.
3. Dictionary
   Dictionary digunakan untuk menyimpan grammar yang menurunkan sebuah terminal, atau pair of variable agar ketika dibutuhkan saat melakukan algoritma CYK pengecekan hanya perlu melihat isi value pada key yang dibutuhkan tidak perlu meng-iterasi seluruh grammar.
4. Tuple
   Tuple digunakan untuk menyimpan pair of variable pada tabel CYK yang merupakan hasil dari cartesian product.

```
def insert_grammar(self, filepath):
```

Fungsi insert_grammar menerima input path sebuah file yang berisi CNF dan menyimpannya pada class CYK.

```
def check_grammar(self, string):
```

Fungsi insert_grammar menerima input sebuah string yang nantinya string akan divalidasi dengan menggunakan grammar yang sudah di masukkan pada class CYK dan akan melakukan parsing dengan menggunakan algoritma CYK.

3.4 Implementasi main.py

Struktur Data

1. List
   List digunakan untuk menyimpan hasil parsing input file yang dipisahkan dengan spasi yang kemudian akan dimodifikasi agar sesuai dengan CNF yang digunakan.

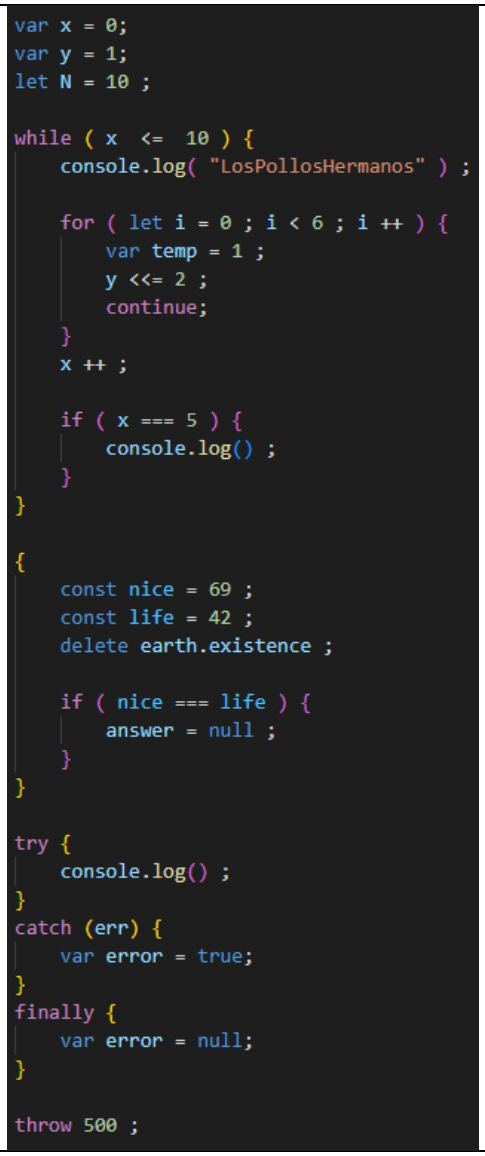
Fungsi dan prosedur yang digunakan pada file ini adalah fungsi dan prosedur yang sudah disebutkan pada sebelumnya.

Antarmuka pada file ini menghasilkan hasil parsing "Accepted" atau "Syntax Error" dilengkapi dengan waktu eksekusi program saat parsing. Kemudian, terdapat pemeriksaan terhadap variabel dengan FA yang sudah dibuat yang akan menghasilkan hasil accepted atau error dengan nama variabel yang menghasilkan error.

3.5 Pengujian Program

No	Analisis	Test Case

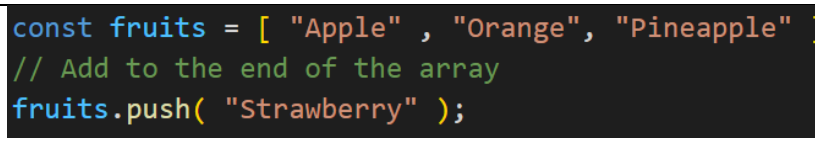
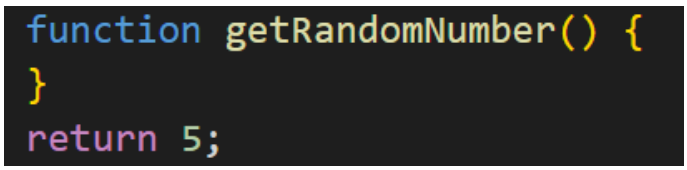
1	• Syntax benar
	Deklarasi fungsi serta if..else di dalamnya tidak bermasalah.
	Return dapat digunakan karena blok kode program berada di dalam fungsi.
	`PS C:\Python\TubesTBFO> python ./src/main.py ./test/test.js`
	`Parsing...`
	`Accepted`
	`Exec time : 10.58`
	`Checking Variable...`
	`No Variable Detected`
	`Exec time : 0.0`
		`// inputAcc.js`
		`function do_something(x){`
		`  // This is a sample comment`
		`  if (x == 0){`
		`  return 0;`
		`  } else if (x + 4 == 1){`
		`  if (true) {`
		`  return 3;`
		`  } else {`
		`  return 2;`
		`  }`
		`  } else if (x == 32){`
		`  return 4;`
		`  } else {`
		`  return "Momen";`
		`  }`
		`}`

2	• Syntax benar
	Deklarasi semua variabel sesuai dengan syntax JS.
	Penggunaan while loop dan for loop di dalamnya sudah benar.
	Statement object disertai method sudah benar.
	Assignment dengan operasi bitwise (y <<= 2) sudah sesuai.
	Penggunaan if sudah sesuai.
	Error handling dengan try, catch, dan finally, serta penggunaan throw exception benar.
	
	`FO> py.exe .\src\main.py .\test\test2.js`
	`Parsing...`
	`Accepted`
	`Exec time : 57.33`
	`Checking Variable...`
	`Variable Name Accepted`
	`Exec time : 0.0`
		

3	• Syntax benar
	Deklarasi variabeel truth dengan nilai boolean true sudah benar.
	Penggunaan switch dan case-casenya telah disertai dengan break, serta penggunaan default pada akhir kondisi.
	Deklarasi variabel _addition dengan penjumlahan dua integer sudah sesuai.
	Penggunaan fungsi subtract dan nilai returnnya yang berupa pengurangan dari sebuah variabel dengan sebuah integer sudah benar.
	`PS C:\Python\TubesTBFO> python ./src/main.py ./test/test3.js`
	`Parsing...`
	`Accepted`
	`Exec time : 5.42`
	`Checking Variable...`
	`Variable Name Accepted`
	`Exec time : 0.0`
		`let truth = true;`
		
		`switch ( truth ) {`
		`  case true:`
		`  console.log(1);`
		`  break;`
		`  case false:`
		`  console.log(0);`
		`  break;`
		`  default:`
		`  break;`
		`}`
		
		`const _addition = 2 + 5;`
		
		`function subtract10(x) {`
		`  return x - 10;`
		`}`

4	• Syntax salah
	kondisi x + 4 == 1 tidak disertai kurung buka dan kurung tutup
	`PS C:\Python\TubesTBFO> python ./src/main.py ./test/test4.js`
	`Parsing...`
	`Syntax Error`
	`Exec time : 12.81`
	`Checking Variable...`
	`No Variable Detected`
	`Exec time : 0.0`
		`function do_something (x) {`
		`  // This is a sample comment`
		`  if (x == 0) {`
		`  return 0;`
		`  } else if x + 4 == 1 {`
		`  if (true) {`
		`  return 3;`
		`  } else {`
		`  return 2;`
		`  }`
		`  } else if (x == 32) {`
		`  return 4;`
		`  } else {`
		`  return "Momen";`
		`  }`
		`}`

5	• Syntax benar
	Penggunaan fungsi serta if..else di dalamnya sesuai.
	Return value yang berupa integer dan perkalian dari sebuah variabel dengan fungsi sudah benar.
	`PS C:\Python\TubesTBFO> python ./src/main.py ./test/test5.js`
	`Parsing...`
	`Accepted`
	`Exec time : 0.32`
	`Checking Variable...`
	`No Variable Detected`
	`Exec time : 0.0`
		`function factorial(x){if(x == 0){ return 1;}else{return x * factorial(x - 1);}}`

6	• Syntax benar Deklarasi dari array dengan menggunakan const sudah sesuai. Pemanggilan method pusg dari objek fruits benar.  	
7	• Syntax salah Return digunakan di luar fungsi.  	

Dari semua pengujian yang sudah dijalankan program parsing yang dibuat dengan CFG yang menggunakan algoritma CYK berhasil mendeteksi sintaks-sintaks dasar dari JS. Namun, program masih relatif lambat seperti yang dapat dilihat pada test case di atas, sehingga memerlukan optimisasi lebih lanjut.

BAB IV

Kesimpulan dan Saran

4.1 Kesimpulan

Setelah melakukan pengerjaan dan pengujian, program parsing *javascript* (node.js) berhasil dibuat sessuai dengan spesifikasi yang sudah diberikan dan dimplementasikan menggunakan materi kuliah yang sudah diajarkan pada mata kuliah IF2124 Teori Bahasa Formal dan Otomata meliputi

1. Finite Automata
2. Context Free Grammar
2. Chomsky Normal Form
3. Algoritma Cocke–Younger–Kasami

4.2 Saran

Melihat program yang sudah dibuat, penulis memiliki beberapa saran menenai Tugas Besar TBFO ini yaitu:

1. Program masih belum menggunakan algoritma yang efisien, contohnya saat melakukan parsing masih perlu disesuaikan dengan CNF sehingga akan menambah banyak character yang perlu dianalisis yang akan menyebabkan waktu CYK akan dengan besar mengingat efisiensi algoritma CYK yang kurang baik.
2. CFG dapat dikembangkan lagi untuk menambahkan sintaks *javascript* yang belum dicakup oleh CFG saat ini

LAMPIRAN

Link repository github.

https://github.com/yansans/TubesTBFO

Pembagian Tugas.

No	Nama	NIM	Tugas
1	Yanuar Sano Nur Rasyid	13521110	Konverter CFG menjadi CNF, FA , Parsing main.py, Laporan
2	Rayhan Hanif Maulana Pradana	13521112	CFG, Test case , Laporan
3	Dewana Gustavus Haraka Otang	13521173	CYK , Main.py , Laporan