

Laporan Tugas Besar
IF2124 Teori Bahasa Formal dan Otomata



Parser Bahasa JavaScript

Disusun oleh:

1. 13518134 Muhammad Raihan Iqbal
2. 13521138 Johann Christian Kandani
3. 13521170 Haziq Abiyyu Mahdy

BAB I

TEORI DASAR

1. *Finite Automata*

Finite Automata (FA) adalah suatu mesin abstrak yang dapat mengenali bahasa reguler. Konsep FA banyak digunakan sebagai model pada perangkat lunak yang digunakan untuk mendesain rangkaian digital serta pada *lexical analyzer* pada *compiler*. FA memiliki himpunan *state* yang berhingga dan dapat berpindah dari satu *state* ke *state* lainnya ketika membaca suatu simbol input. FA memiliki sekumpulan aturan untuk berpindah dari satu *state* ke *state* lain berdasarkan input dan fungsi transisi yang diterapkan. Suatu *string* dapat diterima atau ditolak oleh suatu FA. Suatu *string* $w = a_1a_2...a_n$ diterima oleh suatu FA jika terdapat jalur pada transisi yang dimulai dari *start state*, berakhir di *final state*, dan memiliki sekuens $a_1a_2...a_n$. FA terbagi dua, yaitu *Deterministic Finite Automata* (DFA) dan *Nondeterministic Finite Automata* (NFA). Pada DFA, suatu *state* dan simbol input tidak boleh dipetakan ke lebih dari satu *state* pada fungsi transisi dan tidak diperbolehkan perpindahan null (ϵ), yaitu perpindahan *state* tanpa membaca simbol input. Sedangkan pada NFA, suatu *state* dan simbol input boleh dipetakan ke lebih dari satu *state* pada fungsi transisi dan diperbolehkan perpindahan null (ϵ).

Secara formal, DFA adalah suatu *quintuple* berupa

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

- Q adalah himpunan *state* berhingga
- Σ adalah himpunan simbol input berhingga
- δ adalah fungsi transisi $(q, a) \rightarrow p$ dengan $p, q \in Q$
- q_0 adalah *start state*, $q_0 \in Q$
- F adalah *final state*, $F \subseteq Q$

Sedangkan NFA secara formal adalah suatu *quintuple* berupa

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

- Q adalah himpunan *state* berhingga
- Σ adalah himpunan simbol input berhingga
- δ adalah fungsi transisi $Q \times \Sigma \rightarrow 2^Q$
- q_0 adalah *start state*, $q_0 \in Q$
- F adalah *final state*, $F \subseteq Q$

2. Context Free Grammar (CFG)

CFG atau Context Free Grammar adalah tata bahasa formal di mana setiap aturan produksi adalah dalam bentuk $A \rightarrow B$, dimana A adalah produsen dan B adalah hasil produksi. Batasannya adalah ruas kiri berupa sebuah simbol variabel dan ruas kanan bisa berupa terminal, symbol, variable ataupun ϵ . Contoh aturan produksi yang termasuk CFG adalah seperti berikut ini:

$$X \rightarrow bY \mid Za$$

$$Y \rightarrow aY \mid b$$

$$Z \rightarrow bZ \mid \epsilon$$

CFG adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa.

3. Sintaks JavaScript

Berikut adalah beberapa kata kunci yang harus diimplementasikan dalam *Grammar* pada program ini.

break	const	case	catch	continue
-------	-------	------	-------	----------

default	delete	else	false	finally
for	function	if	let	null
return	switch	throw	try	true
var	while			

Selain kata kunci di atas, diimplementasikan juga *grammar* untuk operasi dan ekspresi dasar seperti penjumlahan, pengurangan, operasi *bitwise*, dsb.

BAB II

HASIL FA DAN CFG

1. Finite Automata (FA)

Pada program ini, digunakan FA untuk mengecek nama variabel serta nilai numerik. FA diimplementasikan menggunakan fungsi di python yang akan menerima suatu string dan melakukan iterasi pada setiap karakter di string. Nilai *state* diinisialisasi dengan 0, dan akan berubah sesuai simbol input yang sedang dibaca dan nilai *state* saat ini. Jika suatu nama variabel atau nilai numerik valid, maka fungsi akan mengembalikan nilai *True*. Sebaliknya, jika suatu nama variabel atau nilai numerik tidak valid, maka fungsi akan mengembalikan nilai *False*.

A. FA untuk nama variabel

Pada bahasa JavaScript, nama variabel harus dimulai dengan huruf abjad (A-Z atau a-z) atau *underscore* ('_'). Karakter selanjutnya dapat diisi dengan huruf abjad, underscore, dan angka (0-9). Berikut adalah tabel transisi FA untuk nama variabel.

States	Uppercase	Lowercase	Underscore	Number (0-9)
→ q0	q1	q1	q1	q2
* q1	q1	q1	q1	q1
q2	--	--	--	--

B. FA untuk nilai numerik

Berikut adalah beberapa kasus nilai numerik yang termasuk valid dalam bahasa JavaScript.

- Nilai numerik yang hanya terdiri atas *integer* (0-9), banyak angka boleh satu atau lebih (contoh; 123 adalah nilai yang valid).
- Nilai numerik yang diawali dengan satu atau lebih *integer*, kemudian diikuti tanda titik('>'). Setelah tanda titik diikuti dengan satu atau lebih

integer, atau *string* boleh berakhir dengan tanda titik, asalkan setelah tanda titik yang pertama, tidak ada tanda titik lagi. (contoh; 29.381 dan 738. adalah nilai yang valid).

- c. Nilai numerik yang diawali tanda titik, kemudian diikuti satu atau lebih integer. (contoh; .493 adalah nilai yang valid). Tanda titik tidak boleh berdiri sendiri atau terdapat lebih dari satu (contoh; ., 4..3, 5.3.0 adalah nilai yang tidak valid).

Berikut adalah tabel transisi FA untuk nilai numerik.

States	Number (0-9)	Point symbol (‘.’)
→ q0	q1	q2
* q1	q1	q3
q2	q3	q4
q3	q3	q4
q4	--	—

2. Context Free Grammar (CFG)

```
# Variables
V -> V1 V2 | A-Z | a-z | _
V1 -> A-Z | a-z | _
V2 -> V2 V2 | A-Z | a-z | 0-9 | _

# Integers
I -> I I1 | 0-9
I1 -> 0-9

# Floats (numeric)
F -> I F1 | F2 I
F1 -> F2 I
F2 -> .

# Booleans
```

```

B -> true | false

# Null
N -> null

# Strings/char
S -> S1 S2 | S4 S5
S1 -> "
S2 -> S3 S1 | "
S3 -> S3 S3 | ASCII (any)
S4 -> '
S5 -> S3 S4 | '

# Operators
# Unary operators (prefix)
O1 -> + | - | ~ | ! | delete

# Binary operators
O2 -> + | - | / | * | % | < | > | & | '|' | ^

# Ternary operators
O3 -> ?

# Expressions
E -> V1 V2 | A-Z | a-z | _ | II1 | 0-9 | IF1 | S1S2 | S5S6 | true |
false | null | E2 E3 | O1 U2 | P U3 | M U4 | V U5 | V U6 | E B1 | E
B2 | E B6 | E B13 | E T1 | A5 E1 | A7 A1

# Expression -> variable / int / float / string / bool (literal)
# E -> V1 V2 | A-Z | a-z | _ | II1 | 0-9 | IF1 | S1S2 | S5S6 | true
| false | null
E1 -> V1 V2 | A-Z | a-z | _ | II1 | 0-9 | IF1 | S1S2 | S5S6 | true
| false | null

# Expression -> (Expression)
# E -> E2 E3
E2 -> (
E3 -> E E4 | )
E4 -> )

# Expressions -> (semua operasi yg ada di bawah)
# E -> O1 U2 | P U3 | M U4 | V U5 | V U6 | E B1 | E B2 | E B6 | E
B13 | E T1 | A5 E1 | A7 A1

# Unary
U -> O1 U2 | P U3 | M U4 | V U5 | V U6

# Unary -> Operator + Operator + ... + Operand (expression)
# U -> O1 U2
U2 -> O1 U2 | V1 V2 | A-Z | a-z | _ | II1 | 0-9 | IF1 | S1S2 | S5S6

```

```

| true | false | null | E2 E3 | O1 U2 | P U3 | M U4 | V U5 | V U6 |
E B1 | E B2 | E B6 | E B13 | E T1 | A5 E1 | A7 A1

# Special case; increment and decrement (e.g. var++, --var)
# U -> P U3 | M U4 | V U5 | V U6
U3 -> P V
P -> +
U4 -> M V
M -> -
U5 -> P P
U5 -> M M

# Binary
B -> E B1 | E B2 | E B6 | E B13

# Binary -> Operand + Operator + Operand + Operator + ... + Operand
# B -> E B1
B1 -> O2 B | O2 E

# >=, <=
# B -> E B2
B2 -> B3 B4
B3 -> > | <
B4 -> B5 E
B5 -> =

# ||, &&, >>, <<, **, ==
# B -> E B6
B6 -> B7 E
B7 -> B8 B8 | B9 B9 | B10 B10 | B11 B11 | B12 B12
B8 -> |
B9 -> &
B10 -> >
B11 -> <
B12 -> *

# ===
# B -> E B13
B13 -> B5 B14
B14 -> B5 B4

# Ternary -> condition ? exprIfTrue : exprIfFalse
T -> E T1
T1 -> O3 T2
T2 -> E T3
T3 -> T4 E
T4 -> :

# Assignment
A -> A5 E1 | A7 A1

```



```

# Assignment -> X = Y = Z = ... (variable) = literal
# A -> A5 E1
A5 -> A5 A5 | V A6
A6 -> =

# +=, -=, *=, /=, %=, |=, &=, ^=, >>=, <<=, ||=, &&=, **=
# A -> A7 A1
A7 -> A7 A7 | V A8
A8 -> A9 A6
A9 -> + | - | / | * | % | & | '|' | ^ | B8 B8 | B9 B9 | B10 B10 |
B11 B11 | B12 B12

CMA -> ,
CLN -> :
SCLN -> ;
CURL1 -> {
CURL2 -> }

# Statement Keyword
## Keyword Control Flow
RTN -> "return"
BRK -> "break"
CNTN -> "continue"
THR -> "throw"
IF -> "if"
ELSE -> "else"
SWTC -> "switch"
CASE -> "case"
DFLT -> "default"
TRY -> "try"
CATCH -> "catch"
FNLY -> "finally"

## Keyword Declaration
VAR -> "var"
LET -> "let"
CONST -> "const"

## Keyword Iterations
WHL -> "while"
FOR -> "for"

# STATEMENTS
STMT -> CURL1 BLK2 | THR E | IF IF2 | SWTC SWTC2 | TRY TRY2 | VAR V
| VAR A | LET V | LET A | CONST V | CONST A | FOR FOR2 | WHL WHL2

STMTSW -> CURL1 BLKSW2 | THR E | IF IFSW2 | SWTC SWTC2 | TRY TRYSW2
| VAR V | VAR A | LET V | LET A | CONST V | CONST A | FOR FOR2 |
WHL WHL2 | BRK

STMTL -> CURL1 BLKL2 | THR E | IF IFL2 | SWTCL SWTCL2 | TRY TRYL2 |

```

```
VAR V | VAR A | LET V | LET A | CONST V | CONST A | FOR FOR2 | WHL  
WHL2 | BRK | CNTN
```

```
# BLOCK
```

```
BLK -> CURL1 BLK2 | CURL1 CURL2
```

```
BLK2 -> START CURL2
```

```
BLKSW -> CURL1 BLKSW2
```

```
BLKSW2 -> STARTSW CURL2
```

```
BLKL -> CURL1 BLKL2
```

```
BLKL2 -> STARTL CURL2
```

```
# Control Flow
```

```
## THROW
```

```
THR -> THR E
```

```
## IF
```

```
IF1 -> IF IF2
```

```
IF2 -> E2 IF3
```

```
IF3 -> E IF4
```

```
IF4 -> E4 IF5 | E4 STMT | E4 E | E4 FCALL
```

```
IF5 -> STMT IF6 | E IF6 | FCALL IF6
```

```
IF6 -> ELSE STMT | ELSE E | ELSE FCALL
```

```
IFSW1 -> IF IFSW2
```

```
IFSW2 -> E2 IFSW3
```

```
IFSW3 -> E IFSW4
```

```
IFSW4 -> E4 IFSW5 | E4 STMTSW | E4 E | E4 FCALL
```

```
IFSW5 -> STMTSW IFSW6 | E IFSW6 | FCALL IFSW6
```

```
IFSW6 -> ELSE STMTSW | ELSE E | ELSE FCALL
```

```
IFL1 -> IF IFL2
```

```
IFL2 -> E2 IFL3
```

```
IFL3 -> E IFL4
```

```
IFL4 -> E4 IFL5 | E4 STMTL | E4 E | E4 FCALL
```

```
IFL5 -> STMTL IFL6 | E IFL6 | FCALL IFL6
```

```
IFL6 -> ELSE STMTL | ELSE E | ELSE FCALL
```

```
## SWITCH
```

```
SWTC1 -> SWTC SWTC2
```

```
SWTC2 -> E2 SWTC3
```

```
SWTC3 -> E SWTC4
```

```
SWTC4 -> E4 SWTC5 | CMA SWTC3
```

```
SWTC5 -> CURL1 SWTC6
```

```
SWTC6 -> CASE SWTC7 | DEFAULT SWTC11
```

```
SWTC7 -> E SWTC8
```

```
SWTC8 -> CLN SWTC9
```

```
SWTC9 -> STARTSW SWTC6 | STARTSW CURL2
```

```
SWTC10 -> CASE SWTC11
```

```
SWTC11 -> E SWTC12
```

```

SWTC12 -> CLN SWTC13
SWTC13 -> STARTSW SWTC10 | STARTSW CURL2

SWTCL1 -> SWTCL SWTCL2
SWTCL2 -> E2 SWTCL3
SWTCL3 -> E SWTCL4
SWTCL4 -> E4 SWTCL5 | CMA SWTCL3
SWTCL5 -> CURL1 SWTCL6
SWTCL6 -> CASE SWTCL7 | DEFAULT SWTCL11
SWTCL7 -> E SWTCL8
SWTCL8 -> CLN SWTCL9
SWTCL9 -> STARTL SWTCL6 | STARTL CURL2
SWTCL10 -> CASE SWTCL11
SWTCL11 -> E SWTCL12
SWTCL12 -> CLN SWTCL13
SWTCL13 -> STARTL SWTCL10 | STARTL CURL2

## TRY - CATCH
TRY1 -> TRY TRY2
TRY2 -> BLK TRY3
TRY3 -> CATCH BLK | CATCH TRY4 | CATCH TRY8 | FNLY BLK
TRY4 -> E2 TRY5
TRY5 -> V TRY6
TRY7 -> E4 BLK | E4 TRY8
TRY8 -> BLK TRY9
TRY9 -> FNLY BLK

TRYSW1 -> TRY TRYSW2
TRYSW2 -> BLKSW TRYSW3
TRYSW3 -> CATCH BLKSW | CATCH TRYSW4 | CATCH TRYSW8 | FNLY BLKSW
TRYSW4 -> E2 TRYSW5
TRYSW5 -> V TRYSW6
TRYSW7 -> E4 BLKSW | E4 TRYSW8
TRYSW8 -> BLKSW TRYSW9
TRYSW9 -> FNLY BLKSW

TRYL1 -> TRY TRYL2
TRYL2 -> BLKL TRYL3
TRYL3 -> CATCH BLKL | CATCH TRYL4 | CATCH TRYL8 | FNLY BLKL
TRYL4 -> E2 TRYL5
TRYL5 -> V TRYL6
TRYL7 -> E4 BLKL | E4 TRYL8
TRYL8 -> BLKL TRYL9
TRYL9 -> FNLY BLKL

# Declaration
DCLR -> VAR V | VAR A | LET V | LET A | CONST V | CONST A

# Iterations
## FOR
FOR1 -> FOR FOR2

```

```

FOR2 -> E2 FOR3 | E2 FOR4
FOR3 -> E FOR4
FOR4 -> SCLN FOR5 | SCLN FOR6
FOR5 -> E FOR6 | DCLR FOR6
FOR6 -> SCLN FOR7 | SCLN FOR8
FOR7 -> E FOR8
FOR8 -> E4 STMTL | E4 E | E4 FCALL

## WHILE
WHL1 -> WHL WHL2
WHL2 -> E2 WHL3
WHL3 -> E WHL4
WHL4 -> E4 STMTL | E4 E | E4 FCALL

# Basic function -> function functionName(params)
FU -> FU1FU2
FU1 -> function
FU2 -> FU3FU4
FU3 -> V1V2 | A-Z | a-z | _
FU4 -> FU5FU6
FU5 -> (
FU6 -> FU7FU10 | FU7FU11
FU7 -> empty | VFU8 | V1V2 | DPFU8 | DP1V | DP1I | DP1S | A-Z | a-z
|
FU8 -> FU9FU7
FU9 -> ,
FU10 -> RPFU11
FU11 -> FU12STMTSW
FU12 -> )

# Arrow function
AF -> AF1AF2 | AF3AF10 | VAF10
AF1 -> async
AF2 -> AF3AF10 | VAF10
AF3 -> AF4AF5
AF4 -> (
AF5 -> AF6AF9
AF6 -> empty | VAF7 | V1V2 | DPAF7 | DP1V | DP1I | DP1S | A-Z | a-z
|
AF7 -> AF8AF6
AF8 -> ,
AF9 -> )
AF10 -> AF11E | AF11STMTSW
AF11 -> =>

# Default params
DP -> DP1V | DP1I | DP1S
DP1 -> VB5

# Rest params
RP -> RP1V

```

```

RP1 -> ...

# const function
CF -> CF1CF2
CF1 -> const
CF2 -> CF3AF
CF3 -> VB5

# Getter
GE -> GE1GE2 | GE1GE3
GE1 -> get
GE2 -> VGE3 | EGE3
GE3 -> GE4STMTSW
GE4 -> GE5GE6
GE5 -> (
GE6 -> )

# Setter
SE -> SE1SE2 | SE1SE3
SE1 -> set
SE2 -> VSE3 | ESE3
SE3 -> SE4STMTSW
SE4 -> SE5SE6
SE5 -> SE7V
SE6 -> )
SE7 -> (

# function usage
FUS -> FUS1FUS2
FUS1 -> V1V2 | A-Z | a-z | _
FUS2 -> FUS3FUS4
FUS3 -> (
FUS4 -> FUS5FUS8 | FUS5FUS9
FUS5 -> empty | VFUS6 | V1V2 | DPFUS6 | DP1V | DP1I | DP1S | A-Z |
a-z | _
FUS6 -> FUS7FUS5
FUS7 -> ,
FUS8 -> RPFUS9
FUS9 -> )

# Class -> class variable {expression}
CL -> CL1CL2
CL1 -> class
CL2 -> VSTMTSW

# Constructor -> CO1CO2 tambahin ke DLCR
CO -> CO1CO2
CO1 -> construtor
CO2 -> CO3CO4
CO3 -> (
CO4 -> CO5CO8

```

```
C05 -> empty | VCO6 | V1V2 | DPCO6 | DP1V | DP1I | DP1S | A-Z | a-z  
|  
C0 $\overline{6}$  -> C07C05  
C07 -> ,  
C08 -> C09STMTSW  
C09 -> )
```

BAB III

IMPLEMENTASI DAN PENGUJIAN

1. Spesifikasi teknis program.

Program akan berbasis *Command Line Interface*. Berikut adalah format yang harus dituliskan pada *command line* untuk menjalankan program

```
python main.py <namaFile.js>
```

Program akan menerima argumen nama file dan menyimpannya pada variabel `args`. Nama file dapat diakses dengan atribut `args.namafile`. Selanjutnya, program akan memanggil fungsi `fileToList` dengan parameter `L`, dengan `L = args.namafile`. Kemudian, program akan memanggil fungsi `CYKParse` untuk list yang telah dihasilkan dari fungsi sebelumnya dan jika teks diterima, akan menampilkan “Accepted” dan jika tidak, akan menampilkan “Syntax Error”.

Header fungsi yang digunakan:

1. `io.py`

```
def fileToList(fileName) :
```

2. `FA.py`

```
def variableFA(s) :  
def numericFA(s) :
```

3. `CYKParser.py`

```
def CYKParse(w) :
```

Penjelasan fungsi-fungsi yang digunakan:

A. `fileToList(fileName)`

Fungsi ini memiliki parameter `fileName` bertipe *string*. Program akan

menemukan file yang bernama `fileName`. Jika file tidak ditemukan, program akan menampilkan kesalahan dan fungsi `fileToList` akan mengembalikan nilai `False`. Jika file ditemukan, maka program akan melakukan iterasi pada seluruh baris yang ada pada file dan memisahkan teks menjadi beberapa kata. Kata tertentu yang berupa *reserved word* akan langsung dimasukkan ke dalam list. Sedangkan kata yang “berkemungkinan” sebagai nama variabel atau nilai numerik akan dimasukkan ke dalam FA untuk mengecek validitas nama variabel atau nilai numerik. Jika valid, maka program akan memasukkan nilai khusus ke dalam list sebagai penanda nama variabel atau nilai numerik untuk memudahkan parsing selanjutnya dengan CYK. Selain dari *reserved word*, nama variabel, dan nilai numerik, program akan memasukkan karakter demi karakter ke dalam list. Setelah proses selesai, fungsi `fileToList` akan mengembalikan list yang telah diisi sebelumnya.

B. `variableFA(s)`

Fungsi ini memiliki parameter `s` bertipe *string*. Program akan melakukan iterasi pada setiap karakter pada *string* dan akan mengembalikan `True` jika string merupakan nama variabel yang valid.

C. `numericFA(s)`

Fungsi ini memiliki parameter `s` bertipe *string*. Program akan melakukan iterasi pada setiap karakter pada *string* dan akan mengembalikan `True` jika string merupakan nilai numerik yang valid.

D. `CYKParse(w)`

Fungsi ini memiliki parameter `w` bertipe *string*. Program akan melakukan *parsing* dengan algoritma Cocke-Younger-Kasami (CYK). Setelah proses selesai, program akan mengembalikan himpunan variabel yang dapat memproduksi `w` (himpunan variabel yang berada pada baris ke `len(w) - 1` dan kolom ke 0 pada tabel CYK).

Struktur data yang digunakan

1. `rules.py`

```
# Terminals
uppercase=[chr(ord('A')+i) for i in range(26)]
```



```

lowercase=[chr(ord('a')+i) for i in range(26)]
integer=[chr(ord('0')+i) for i in range(10)]
nonzero=[chr(ord('1')+i) for i in range(9)]
binary=["+", "-", "*", "/", "%", "|", "^", "&"]
boolop=["and", "or", "is", "in", "<", ">"]
unary=["Not", "+", "-", "~"]
boolean=["True", "False", "None"]

# Rules of the grammar
R = {
    # Variables
    "V": ["V1", "V2"] + uppercase + lowercase + ['_'],
    "V1": uppercase + lowercase + ['_'],
    "V2":
["V2", "V2"] + uppercase + lowercase + integer + ['_'],
    .
    .
    .

    # basic function
    "FU": ["FU1", "FU2"],
    "FU1": ["function"],
    "FU2": ["FU3", "FU4"],
    "FU3": ["V1", "V2"] + uppercase + lowercase + ['_'],
    "FU4": ["FU5", "FU6"],
    "FU5": ["("],
    "FU6": ["FU7", "FU10"], ["FU7", "FU11"],
    "FU7": ["empty"], ["V", "FU8"], ["V1", "V2"], ["DP",
"FU8"], ["DP1", "V"], ["DP1", "I"], ["DP1",
"S"] + uppercase + lowercase + ['_'],
    "FU8": ["FU9", "FU7"],
    "FU9": [","],
    "FU10": ["RP", "FU11"],
    "FU11": ["FU12", "STMTSW"],
    "FU12": [")"],
    .
    .
    .

    # const function
    "CF": ["CF1", "CF2"],
    "CF1": ["const"],
    "CF2": ["CF3", "AF"],
    "CF3": ["V", "B5"],
    .
    .
    .

    # Class: class variable {expression}
    "CL": ["CL1", "CL2"],
    "CL1": ["class"],
    "CL2": ["V", "STMTSW"],
    .

```

.

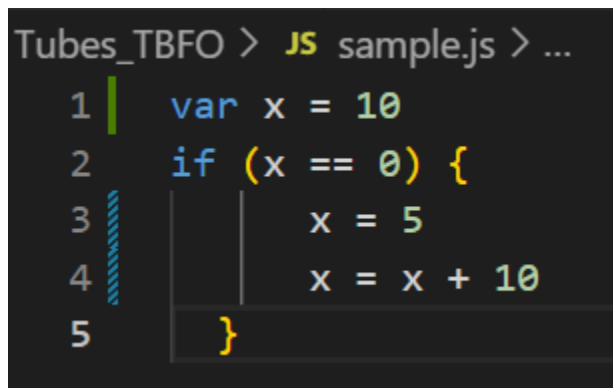
Penjelasan struktur data yang digunakan:

Secara umum, struktur data yang digunakan berisi tipe data primitif, list, *string*, serta *dictionary* yang berisi rule-rule produksi dalam bentuk Chomsky Normal Form (CNF). Bentuk CNF ini dipilih agar dapat langsung digunakan pada algoritma CYK yang telah diimplementasikan. Bagian *key* pada *dictionary* merupakan variabel yang berada pada ruas kiri pada *production*, sedangkan bagian *value* pada *dictionary* merupakan variabel atau terminal yang berada pada ruas kanan pada *production*. Berikut adalah ilustrasi konversi *production rules* pada CFG menjadi *dictionary* di python.

```
# Production rule
I -> I1I2 | I2I2
I1 -> a | b | c
I2 -> d | e

# Python dictionary
Rules = {
    "I" : [["I1", "I2"], ["I2", "I2"]],
    "I1" : [["a"], ["b"], ["c"]],
    "I2" : [["d"], ["e"]]
}
```

2. Kasus uji



```
Tubes_TBFO > JS sample.js > ...
1 | var x = 10
2 | if (x == 0) {
3 |     x = 5
4 |     x = x + 10
5 | }
```

```
PS D:\University\SMT III\IF2124 Teori Bahasa Formal dan Otomata\Tubes_TBFO\src> python main.py ../sample.js
Accepted
PS D:\University\SMT III\IF2124 Teori Bahasa Formal dan Otomata\Tubes_TBFO\src> 
```

JS sample.js X

JS sample.js > ...

```
1
2    // This is a sample comment
3    if (x == 0) {
4        var a = 0;
5    } else if (x + 4 == 1) {
6        if (true) {
7            let b = 3;
8        } else {
9            const a = 2;
10        }
11    } else if (x == 32) {
12        a = 4;
13    } else {
14        x == 2
15        "Momen";
16    }
17
18
19
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Writing objects: 100% (6/6), 616 bytes | 616.00 KiB/s, done.
Total 6 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/haziqam/Tubes_TBFO
c9ae414..c0aad7 main -> main
PS C:\Users\Johann\Documents\Code\Kuliah\3\TBFO\Tubes_TBFO\src> py main.py ..\sample.js
Accepted
PS C:\Users\Johann\Documents\Code\Kuliah\3\TBFO\Tubes_TBFO\src> |

JS sample.js

JS fail.js 1, U X

JS fail.js > ...

```
1
2 // This is a sample comment
3 if (x == 0) {
4   var a = 0;
5 } else if (x + 4 == 1) {
6   if (true) {
7     let b = 3;
8   } else {
9     break;
10  }
11 } else {
12   x == 2
13   "Momen";
14
15 }
16
17
18
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
remote: Resolving deltas: 100% (5/5), completed with 5 local objects.
To https://github.com/haziqam/Tubes_TBFO
   c9ae414..c0aad7  main -> main
PS C:\Users\Johann\Documents\Code\Kuliah\3\TBFO\Tubes_TBFO\src> py main.py ..\sample.js
Accepted
PS C:\Users\Johann\Documents\Code\Kuliah\3\TBFO\Tubes_TBFO\src> py main.py ..\fail.js
Syntax Error
PS C:\Users\Johann\Documents\Code\Kuliah\3\TBFO\Tubes_TBFO\src> 
```

BAB IV

LINK REPOSITORY DAN PEMBAGIAN TUGAS

Link repository:

https://github.com/haziqam/Tubes_TBFO

Pembagian tugas:

Nama	Bagian yang dikerjakan
Muhammad Raihan Iqbal	variableFA pada FA.py, rules pada rules.py, CFG <i>function</i> dan <i>class</i> .
Johann Christian Kandani	CYKParse.py, CFG <i>statement</i> , <i>statement</i> pada rules.py
Haziq Abiyyu Mahdy	io.py, numericFA pada FA.py, main.py, CFG <i>expression</i> dan <i>operation</i> .