

**LAPORAN TUGAS BESAR**  
**IF2124 TEORI BAHASA FORMAL DAN OTOMATA**  
**COMPILER BAHASA JAVASCRIPT**



**Disusun oleh :**

Muhammad Abdul Aziz Ghazali - 13521128

Muhammad Zaki Amanullah - 13521146

Mohammad Rifqi Farhansyah – 13521166

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG BANDUNG**

# DAFTAR ISI

DAFTAR ISI.....	2
DESKRIPSI MASALAH.....	3
TEORI DASAR.....	4
2.1 Context Free Grammar.....	4
2.2 CHOMSKY NORMAL-FORM.....	4
2.3 COCKE-YOUNGER KASAMI.....	5
2.4 Bahasa Javascript dan Node.js .....	5
2.5 Bahasa Python .....	6
IMPLEMENTASI DASAR .....	7
3.1 Implementasi CFG .....	7
IMPLEMENTASI DAN PENGUJIAN.....	12
4.1 Spesifikasi Teknis Program .....	12
4.2 Eksperimen .....	14
KESIMPULAN DAN SARAN .....	17
5.1 Kesimpulan .....	17
5.2 Saran .....	17
REFERENSI .....	18

# BAB I

## DESKRIPSI MASALAH

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks bahasa atau parsing yang dibuat oleh programmer untuk memastikan program dapat dieksekusi tanpa menghasilkan *error*. Parsing 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 parsing. Sudah sangat banyak grammar dan algoritma yang dikembangkan untuk menghasilkan compiler dengan performa yang tinggi. Terdapat *CFG*, *CNF<sub>-e</sub>*, *CNF<sub>+e</sub>*, *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 program ini diimplementasikan parser untuk *JavaScript (Node.js)* untuk beberapa statement dan sintaks bawaan *JavaScript*. Konsep *CFG* digunakan untuk pengerjaan parser yang mengevaluasi syntax program. *FA* digunakan untuk nama variabel dan operasi (+, -, >, dll) dalam program.

Algoritma yang dipakai adalah algoritma *CYK (Cocke-Younger-Kasami)*. Algoritma *CYK* menggunakan grammar *CNF (Chomsky Normal Form)* sebagai grammar masukannya. Oleh karena itu, grammar dalam *CFG (Context Free Grammar)* harus dibuat terlebih dahulu, kemudian grammar *CFG* tersebut dikonversikan ke grammar *CNF*. Konsep *CYK*, *CNF*, dan *CFG* sudah dipelajari di dalam mata kuliah Teori Bahasa Formal dan Automata.

# BAB II

## TEORI DASAR

### 2.1 Context Free Grammar

CFG atau *Context Free Grammar* adalah tata bahasa formal di mana setiap aturan produksi adalah dalam bentuk  $A \rightarrow B$  di mana  $A$  adalah pemproduksi, dan  $B$  adalah hasil produksi. Batasannya hanyalah ruas kiri adalah sebuah simbol variabel. Dan pada ruas kanan bisa berupa terminal, *symbol*, *variable* ataupun  $\epsilon$ , 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. Definisi formal dari CFG dapat didefinisikan sebagai berikut.

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

$V$  = himpunan terbatas variabel

$T$  = himpunan terbatas terminal

$P$  = himpunan terbatas dari produksi

$S$  = start symbol

CFG menjadi dasar dalam pembentukan suatu proses analisis secara sintaksis. Bagian sintaks dalam suatu *compiler* kebanyakan didefinisikan dalam CFG. Pohon penurunan (*parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi terminal. Setiap dari simbol variabel akan diturunkan menjadi terminal sampai tidak ada lagi yang masih variabel.

Proses *parsing* tersebut bisa dilakukan sebagai berikut:

- *Leftmost Derivation*, yaitu simbol variabel yang paling kiri diganti terlebih dahulu.
- *Rightmost Derivation*, yaitu simbol variabel yang paling kanan diganti terlebih dahulu.

### 2.2 CHOMSKY NORMAL-FORM

*Chomsky Normal Form* atau biasa disebut sebagai CNF merupakan salah satu bentuk normal dari *Context Free Grammar* (CFG). CNF dapat dibuat dari CFG dengan melalui tahap penyederhanaan yaitu penghilangan produksi yang “tidak penting”, unit, dan  $\epsilon$ . Sehingga, CNF memiliki syarat yaitu tidak memiliki produksi useless, tidak memiliki produksi unit, dan tidak memiliki  $\epsilon$ . Bentuk dari CNF produksi CNF adalah sebagai berikut.

$$A \rightarrow BC \text{ atau } A \rightarrow a$$

$A = \text{variabel}$

$B = \text{terminal}$

Untuk membuat sebuah CNF terdapat langkah-langkah pembentukannya yang secara umum dapat dideskripsikan sebagai berikut.

- Biarkan aturan produksi yang sudah dalam bentuk CNF.
- Penggantian aturan produksi yang ruas kanannya memuat symbol teriman dan dengan panjang ruas kanannya lebih dari satu.
- Penggantian aturan produksi yang ruas kanannya memuat lebih dari dua simbol variabel.
- Penggantian tersebut dilakukan berkali-kali sampai pada akhirnya aturan produksi memenuhi aturan dari CNF.

### 2.3 COCKE-YOUNGER KASAMI

*Cocke-Younger Kasami* atau biasa disebut sebagai CYK merupakan salah satu algoritma *parsing* yang digunakan dalam CFG. Untuk dapat menggunakan CYK ke dalam sebuah *grammar*, bentuk dari *grammar* tersebut harus dalam bentuk CNF. Algoritma menggunakan *dynamic programming* untuk menentukan apakah suatu string termasuk ke dalam sebuah *grammar*.

Sebagai contoh diberikan aturan produksi sebagai berikut :

$$S \rightarrow AB \mid BC$$
$$A \rightarrow BA \mid a$$
$$B \rightarrow CC \mid b$$
$$C \rightarrow AB \mid a$$

### 2.4 Bahasa Javascript dan Node.js

JavaScript adalah bahasa pemrograman yang digunakan dalam pengembangan website agar lebih dinamis dan interaktif. JavaScript dapat meningkatkan fungsionalitas pada halaman web. Bahkan dengan JavaScript ini kamu bisa membuat aplikasi, tools, atau bahkan game pada web. Versi awal bahasa JS hanya dipakai di kalangan Netscape beserta dengan fungsionalitas pun yang masih terbatas. Singkat cerita pada tahun 1996 JavaScript secara resmi dinamakan sebagai ECMAScript. ECMAScript 2 dikembangkan pada tahun 1998 yang dilanjutkan dengan ECMAScript 3 setahun kemudian. Node.js adalah runtime environment untuk JavaScript yang bersifat open-source dan cross-platform. Dengan Node.js kita dapat menjalankan kode JavaScript di mana pun, tidak hanya terbatas pada lingkungan browser. Node.js menjalankan V8 JavaScript engine (yang juga merupakan inti dari Google Chrome) di luar browser. Ini memungkinkan Node.js memiliki performa yang tinggi.

## **2.5 Bahasa Python**

Python adalah bahasa interpreter tingkat tinggi (high level), dan juga general-purpose. Python dibuat 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 berbagai paradigma yaitu fungsional, imperatif, berorientasi objek, dan reflektif. Bahasa ini kami gunakan untuk software ini. Beberapa alasannya adalah karena librarnya yang luas dan penggunaannya yang mudah serta dynamically typed.

# BAB III

## IMPLEMENTASI DASAR

### 3.1 Implementasi CFG

Berikut adalah Context-Free Grammar untuk membuat parser bahasa python yang telah kami buat.

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

- **Non-Terminal Symbol (V)** = { S, STMTS, NL, STMT, COND, LOOP, ASSIGN, IMPORT, FUNCTION, TRY, CONST, VAR, LET, THROW, SWITCH, STMTLOOP, BREAK, CONT, EXP, PAREXP, DOTEXP, LOGICEXP, MATHEXP, FUNCEXP, ANONFUNCTION, ARR, ARREXP, ELMT, DICT, TUPLE, LIT, DOTEXPR, ASSIGN, ASSIGNCHAIN, ASSIGNCHAIN1, ASSIGNOP, COMMA, CONT, SWITCH, LP, LC, RP, RC, CASES, DEFAULT, CASE, LP, LIT, STMTLOOP, SC, DEFAULT, CONST, VAR, LET, LOGICEXP, RELOP, MATHEXP, MATHOP1, MATHOP2, FUNCEXP, TUPLE, THROW, LC, RC, CF, TRY, CF, FINALLY, ARR, LB, RB, SC, ARREXP, ARRLIST, ELMT, RB, LB, PARAM, OBJ, OBJ1, OBJLIST, OBJECTLITERAL, OBJECTBINDINGPATTERN, ARGS, REAL, IMPORT, DEFWC, PKGS, EXPORTS, EXPORT, EXPORTAS, COND, CONDTAIL, DELETE, LOOP, WITH, RET, ID, ID1, FUNCTION, ANONFUNCTION, CLASS, INHERIT, COMMA, COLON, LC, RC, LB, RB, LP, RP, NL, SC }
- **Terminal Symbol (T)** = { 'id', 'int', 'str', 'dot', 'eq', 'pluseq', 'mineq', 'diveq', 'modeq', 'poweq', 'ampeq', 'boreq', 'xoreq', 'sreq', 'usreq', 'sleq', 'andeq', 'oreq', 'nullishCoalescingEq', 'inc', 'dec', 'break', 'continue', 'switch', 'case', 'default', 'const', 'var', 'let', 'and', 'or', 'in', 'not', 'neq', 'equal', 'strictEqual', 'gt', 'gte', 'lt', 'lte', 'plus', 'min', 'bnot', 'mult', 'pow', 'div', 'mod', 'sl', 'sr', 'usr', 'amp', 'bor', 'xor', 'throw', 'try', 'catch', 'finally', 'null', 'true', 'false', 'xbo', 'import', 'from', 'as', 'lc', 'rc', 'comma', 'default', 'if', 'question', 'else', 'delete', 'while', 'with', 'for', 'in', 'function', 'class', 'colon', 'return' }
- **Productions (P)**

Production	Result
SS	STMTS NL
STMTS	STMTS NL STMT   STMTS SC STMT   STMT
STMT	COND   LOOP   ASSIGN   IMPORT   FUNCTION   TRY   CONST   VAR   LET   THROW   SWITCH
STMTLOOP	STMTLOOP STMTS   STMTLOOP BREAK   STMTLOOP CONT   STMTS   BREAK   CONT

EXP	PAREXP   DOTEXP   LOGICEXP   MATHEXP   FUNCEXP   ANONFUNCTION   ARR   ARREXP   ELMT   DICT   TUPLE   LIT   id   int   str
DOTEXP	EXP dot DOTEXPR   EXP optChain DOTEXPR
DOTEXPR	FUNCEXP   ELMT   id
ASSIGN	ASSIGNCHAIN   id ASSIGNOP EXP
ASSIGNCHAIN	id eq ASSIGNCHAIN COMMA   id eq EXP SC   id eq EXP
ASSIGNCHAIN1	id eq ASSIGNCHAIN1 COMMA   id eq EXP   id   id eq EXP SC   id SC
ASSIGNOP	eq   pluseq   mineq   multeq   diveq   modeq   poweq   ampeq   boreq   xoreq   sreq   usreq   sleq   andeq   oreq   nullishCoalescingEq   id inc   id dec   inc id   dec id
BREAK	break SC   break
CONT	continue SC   continue
SWITCH	switch LP EXP RP LC CASES RC
CASES	CASE   CASE DEFAULT CASE   DEFAULT CASE   CASE DEFAULT   DEFAULT
CASE	case LP LIT RP COLON STMTLOOP BREAK SC CASE   case LP LIT RP COLON STMTLOOP BREAK SC   case LP LIT RP COLON STMTLOOP BREAK CASE   case LP LIT RP COLON STMTLOOP BREAK
DEFAULT	default COLON STMTLOOP BREAK
CONST	const ASSIGNCHAIN SC   const ASSIGNCHAIN
VAR	var ASSIGNCHAIN SC   var id SC   var ASSIGNCHAIN   var id
LET	let ASSIGNCHAIN SC   let id SC   let ASSIGNCHAIN   let id
LOGICEXP	EXP RELOP EXP   not EXP
RELOP	and   or   nullishCoalescing   in   not   neq   equal   strictEqual   gt   gte   lt   lte
MATHEXP	EXP MATHOP2 EXP   MATHOP1 EXP
MATHOP1	plus   min   bnot



MATHOP2	mult   pow   div   mod   plus   min   sl   sr   usr   amp   bor   xor
FUNCEXP	id TUPLE
THROW	throw EXP SC   throw EXP
TRY	try LC STMTS RC CF   try LC RC CF
CF	CATCH   FINALLY   CATCH FINALLY
CATCH	catch LP PARAM RP LC STMTS RC   catch LP PARAM RP LC RC
FINALLY	finally LC STMTS RC   finally LC RC
ARR	LB EXP RB SC   LB RB SC   LB EXP RB   LB RB
ARREXP	id LB ARRLIST RB SC   id LB ARRLIST RB
ELMT	id LB EXP RB
PARAM	id   id COMMA PARAM
OBJ	LP OBJ RP   LC OBJLIST RC   LC RC
OBJLIST	OBJLIST OBJ1   OBJ1
OBJ1	str COLON EXP COMMA   str COLON FUNCEXP
TUPLE	LP ARGS RP   LP RP
ARGS	ARGS COMMA EXP   EXP
LIT	REAL   int   xbo   str   false   true   null
REAL	int dot int   dot int
IMPORT	import PKGS from str SC   import DEFWC as id from str SC   import str SC
DEFWC	id wildcard   id
PKGS	EXPORTS   DEFWC
EXPORTS	lc EXPORT rc   lc EXPORT rc comma DEFWC   DEFWC comma lc EXPORT rc   lc EXPORTAS rc   lc str rc   lc default rc
EXPORT	id comma EXPORT   id
EXPORTAS	id comma EXPORTAS comma id   str comma EXPORTAS comma id   id as id   str as id
COND	if LP EXP RP LC STMTS RC CONDTAIL   if LP LIT RP LC STMTS RC CONDTAIL   ID1 eq EXP question EXP COLON EXP SC   ID1 eq EXP question EXP COLON EXP S   if LP EXP RP LC STMTS RC   if LP LIT RP LC STMTS RC   if LP EXP RP LC RC CONDTAIL   if LP

	LIT RP LC RC CONDTAIL   ID1 eq EXP question EXP COLON EXP SC   ID1 eq EXP question EXP COLON EXP S   if LP EXP RP LC RC   if LP LIT RP LC RC
CONDTAIL	else if LP EXP RP LC STMTS RC CONDTAIL   CONDTAIL else LC STMTS RC   else if LP EXP RP LC STMTS RC   else LC STMTS RC   else if LP EXP RP LC RC CONDTAIL   CONDTAIL else LC RC   else if LP EXP RP LC RC   else LC RC
DELETE	delete EXP SC   delete SC   delete EXP   delete
LOOP	while LP EXP RP LC STMTLOOP RC   for LP EXP in EXP RP LC STMTLOOP RC   for LP ASSIGNCHAIN SC RELOP SC MATHEXP RP LC STMTLOOP RC   for LP id SC RELOP SC MATHEXP RP LC STMTLOOP RC
WITH	with EXP as id COLON STMTS
RET	return EXP SC   return SC   return EXP   return
ID	id   ID dot id
ID1	let id   var id   id
FUNCTION	function id LP PARAM RP LC STMTS RC SC   function id LP RP LC STMTS RC SC   function id LP PARAM RP LC STMTS RC   function id LP RP LC STMTS RC   function id LP PARAM RP LC RC SC   function id LP RP LC RC SC   function id LP PARAM RP LC RC   function id LP RP LC RC
ANONFUNCTION	function LP PARAM RP LC STMTS RC SC   function LP RP LC STMTS RC SC   function LP PARAM RP LC STMTS RC   function LP RP LC STMTS RC   function LP PARAM RP LC RC SC   function LP RP LC RC SC   function LP PARAM RP LC RC   function LP RP LC RC
CLASS	class id LP INHERIT RP COLON S   class id COLON S   class id LP

	INHERIT RP COLON S   class id LP RP COLON S   class id LP RP COLON S	
INHERIT	INHERIT COMMA PKG   PKG   COMMA PKG	
COMMA	comma NL	comma
COLON	colon NL	colon
LC	lc NL	lc
RC	NL rc	rc
LB	lb NL	lb
RB	NL rb	rb
LP	lp NL	lp
RP	NL rp	rp
NL	NL nl	nl
SC	sc NL	sc

- **Start Symbol (S) = S**

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Spesifikasi Teknis Program

a. Grammar.txt

File txt berisikan grammar bahasa pemrograman javascript yang akan diparse untuk digunakan dalam algoritma CYK.

b. grammar\_parser.py

i. function loadGrammar(grammarPath) -> (terminals, variables, productions)

Fungsi yang menerima sebuah path yang terdirikan grammar bahasa javascript yang digunakan. Fungsi ini akan mengembalikan triple yang berisikan list terminal, list variabel, dan list produksi dari CFG yang telah di-parse.

c. grammar\_convert.py

i. function generatingProduction(productions, terminals, variables) -> Dictionary

Fungsi ini menerima list produksi, list terminal serta list variabel. Selanjutnya fungsi ini akan menghasilkan sebuah dictionary yang berisikan semua produksi dari CFG yang “generating” atau produksi yang menghasilkan tepat satu buah terminal. Semua produksi tersebut nantinya akan disimpan dalam sebuah dictionary dengan key berupa semua terminal dan value berupa variabel yang menghasilkan terminal tersebut.

ii. function isUnitProduction(prod, variables) -> boolean

Fungsi ini akan menerima dua buah parameter berupa list produksi dan list variabel dan akan mengembalikan sebuah boolean yang akan menyatakan apakah produksi tersebut merupakan “Unit Production” atau bukan. Unit production adalah produksi yang menghasilkan hanya satu buah variabel non terminal.

iii. function isSimpleForm(t, var, prod) -> boolean

Fungsi ini mengembalikan nilai boolean yang menyatakan apakah sebuah produksi *prod* merupakan produksi dalam simple form atau bukan. Produksi dalam simple form adalah produksi yang tepat menghasilkan satu buah terminal.

iv. function removeUnitProduction(productions, variables) -> List

Fungsi ini akan menghasilkan list production baru yang telah mengeliminasi semua unit production dari *production* awal.

v. function productionToDictionary(productions) -> Dictionary

Fungsi ini menghasilkan dictionary hasil konversi list produksi dari CFG atau CNF.

vi. function convertToCNF(productions, terminals, variables) -> List

Fungsi ini akan menghasilkan list yang berisikan rules dari sebuah CNF yang merupakan hasil konversi CFG yang telah di parse dari file txt. Fungsi ini menerima 3 buah parameter, yaitu productions yang berisikan daftar produksi dari CFG, terminals yang berisikan daftar terminal CFG, dan variables yang berisikan daftar variabel dari CFG.

Fungsi ini tidak mempertimbangkan adanya eliminasi epsilon karena semua produksi yang menghasilkan epsilon sudah ditangani secara manual. Fungsi ini pertama-tama akan menghapus semua unit production dari produksi CFG dengan mengiterasikan fungsi removeUnitProduction yang telah dibuat sebelumnya hingga produksi inisial dan produksi akhir merupakan produksi yang sama yang menandakan bahwa semua unit production pada grammar sudah dieliminasi.

Fungsi ini juga mengasumsikan bahwa semua produksi dapat tercapai dari start production dan semua production merupakan production yang generating atau produksi yang akan dapat menghasilkan terminal.

Langkah selanjutnya adalah untuk mengubah semua bentuk produksi menjadi  $A \rightarrow AB$  dan  $A \rightarrow a$  dengan  $A$  dan  $B$  adalah sebuah variabel dan  $a$  adalah sebuah terminal.

vii. function displayCNF(rules) -> string

Fungsi ini menerima daftar rules pada sebuah CNF lalu akan menkonversi daftar rules tersebut menjadi sebuah dictionary. Setelah itu dihasilkan sebuah string yang akan di concatenate dengan sebuah variabel atau terminal yang sesuai.

d. token.py

i. function lexxer()

Fungsi menerima code javascript yang sudah dijadikan sebuah string yang panjang. kemudian string tersebut akan dicocokkan dengan pola regex untuk menemukan token yang bersesuaian dengan isi text. Token-token disimpan dalam list of string. Juga dilakukan validasi karakter dalam text yang legal dalam javascript dan jika salah akan diberikan tahu lokasi kesalahan.

List token akan di return oleh fungsi ini.

ii. Function createToken()

Fungsi menerima string berupa alamat file code javascript yang akan diproses. Dari alamat file akan diakses file tersebut dan diubah ke string yang akan digunakan sebagai input fungsi lexxer. Hasil token juga disimpan dalam file txt.

e. Main.py

i. procedure banner()

Fungsi ini merupakan fungsi untuk display UI pada program yang akan digunakan.

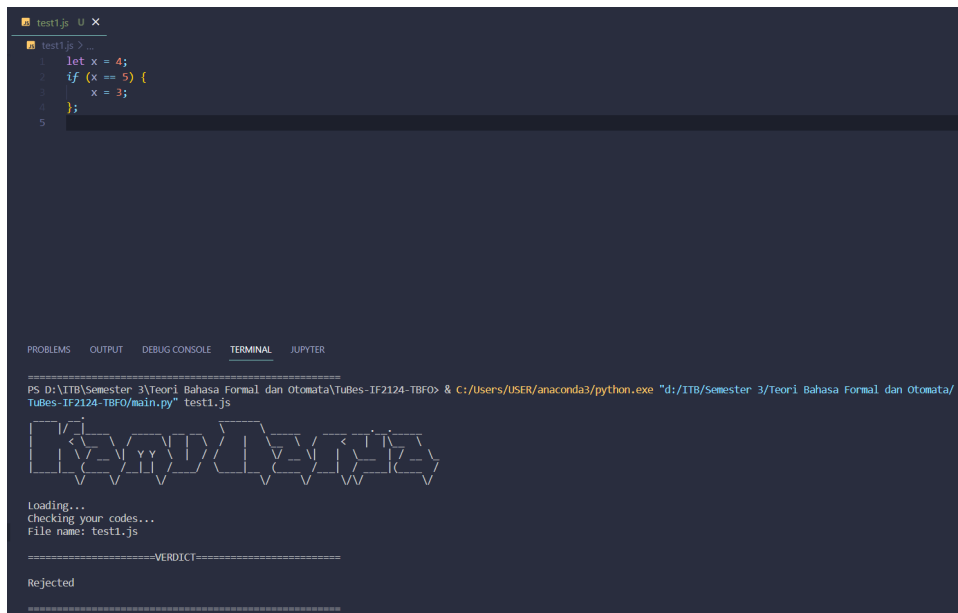
ii. function verdict()

Fungsi utama yang akan dijalankan ketika program berjalan. Digunakan library `argparse` untuk parsing nama file yang akan digunakan pada runtime. Lalu akan dihasilkan sebuah token berdasarkan file tersebut dengan fungsi `createToken()` yang sudah direalisasikan di file `token.py`. Setelah itu dilakukan loading dan konversi grammar menjadi CNF. Setelah itu dilakukan konversi grammar dari tipe data list menjadi tipe data dictionary agar dapat diimplementasikan algoritma CYK kepadanya. Lalu dilakukan algoritma CYK pada dictionary tersebut dan token yang sudah didapatkan. Hasil dari algoritma CYK akan ditampilkan.

## 4.2 Eksperimen

### 1. Test case 1

Syntax javascript yang sederhana yang seharusnya menghasilkan verdict yang accepted.



```
test1.js U X
test1.js > ...
let x = 4;
if (x == 5) {
  x = 3;
};
5

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

p5 D:\ITB\Semester 3\Teori Bahasa Formal dan Otomata\Tubes-IF2124-TBFO> & C:/Users/USER/anaconda3/python.exe "d:/ITB/Semester 3/Teori Bahasa Formal dan Otomata/
Tubes-IF2124-TBFO/main.py" test1.js

Loading...
Checking your codes...
File name: test1.js

=====VERDICT=====
Rejected
```

### 2. Test case 2 (rejected)

Test case dengan menggunakan bahasa pemrograman lain akan menghasilkan verdict yang rejected.

```
1 #include <iostream>
2
3 int main() {
4     cout << "Hello world" << endl;
5     return 0;
6 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

Rejected

PS D:\ITB\Semester 3\Teori Bahasa Formal dan Otomata\TuBes-IF2124-TBFO> & C:/Users/USER/anaconda3/python.exe "d:/ITB/Semester 3/Teori Bahasa Formal dan Otomata/TuBes-IF2124-TBFO/main.py" test2.jsnormal dan Otomata\TuBes-IF2124-TBFO>

Loading...

Checking your codes...

File name: test2.js

SYNTAX ERROR

Illegal character # at line 1 and column 1

### 3. Test case 3

Test case dengan syntax yang sesuai dengan ketentuan javascript.

```
1 function do something(x) {
2     // This is a sample comment
3     if (x == 0) {
4         return 0;
5     } else if (x + 4 == 1) {
6         if (true) {
7             return 3;
8         } else {
9             return 2;
10        }
11    } else if (x == 32) {
12        return 4;
13    } else {
14        return "Homen";
15    }
16 }
17
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

PS D:\ITB\Semester 3\Teori Bahasa Formal dan Otomata\TuBes-IF2124-TBFO> & C:/Users/USER/anaconda3/python.exe "d:/ITB/Semester 3/Teori Bahasa Formal dan Otomata/TuBes-IF2124-TBFO/main.py" test3.js

Loading...

Checking your codes...

File name: test3.js

VERDICT

Rejected


### 4. Test case 4 (rejected)

Test case dengan syntax yang tidak sesuai (tidak ada parantheses yang mengapit conditional expression).

```
test4.js 2, U x
1 function do_something(x) {
2   // this is a sample multiline comment
3   if (x == 0) {
4     return 0;
5   } else if (x + 4 == 1) {
6     if (true) {
7       return 3;
8     } else {
9       return 2;
10    }
11  }
12  } else if (x == 32) {
13    return 4;
14  } else {
15    return "Momen";
16  }
17 }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

PS D:\ITB\Semester 3\Teori Bahasa Formal dan Otomata\TuBes-IF2124-TBFO> & C:/Users/USER/anaconda3/python.exe "d:/ITB/Semester 3/Teori Bahasa Formal dan Otomata/TuBes-IF2124-TBFO/main.py" test4.js



Loading...  
Checking your codes...  
File name: test4.js

=====VERDICT=====

Rejected



## BAB V

### KESIMPULAN DAN SARAN

#### 5.1 Kesimpulan

- 5.1.1 Program dapat menerima suatu teks file atau string dan melakukan evaluasi kebenaran kode tersebut dengan memanfaatkan CFG dan FA.
- 5.2.1 *Grammar* yang dimiliki oleh bahasa *JavaScript* sangatlah luas. *Grammar* yang kami buat belum dapat menangani keseluruhan *grammar* yang dimiliki oleh *JavaScript*.

#### 5.2 Saran

- 5.2.1 Dilakukan pengembangan *grammar* agar program yang dibuat semakin menyerupai *compiler JavaScript*.
- 5.2.2 Melakukan banyak eksplorasi mengenai Finite Automata, Context-Free Grammar, dan algoritma CYK agar program dapat berjalan dengan lebih efektif dan efisien.

## REFERENSI

- Rockbutton. 2020. *Markdown-like DSL for defining grammatical syntax for programming languages*. Diakses pada 25 November 2022, dari <https://github.com/rbuckton/grammarkdown>.
- MDN Contributors. 2022. *JavaScript – Programming Languages*. Diakses pada 24 November 2022, dari <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- Waldemar, H. 1999. *JavaScript Grammar 2.0 & 1.4*. Diakses pada 24 November 2022, dari <https://www-archive.mozilla.org/js/language/grammar14.html#N-Expression>.