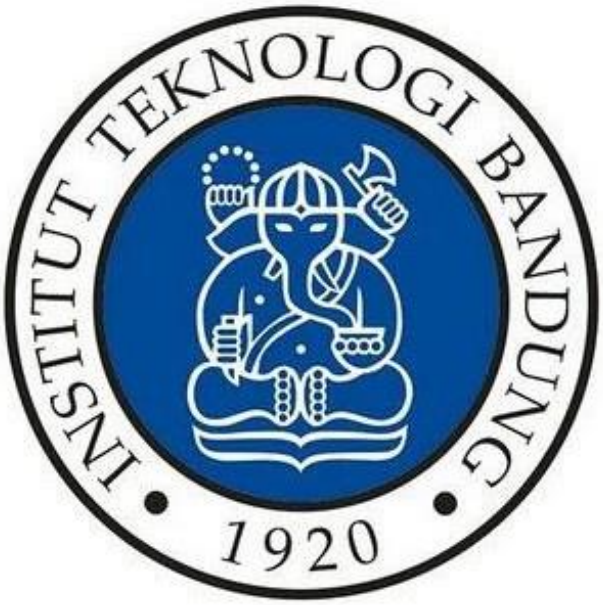
IF2124 Teori Bahasa Formal dan Otomata

Laporan Tugas Besar

Parser Bahasa JavaScript



Kelompok : GaruParsing

Nigel Sahl - 13521043
Ghazi Akmal Fauzan - 13521058
Muhammad Fadhil Amri - 13521066

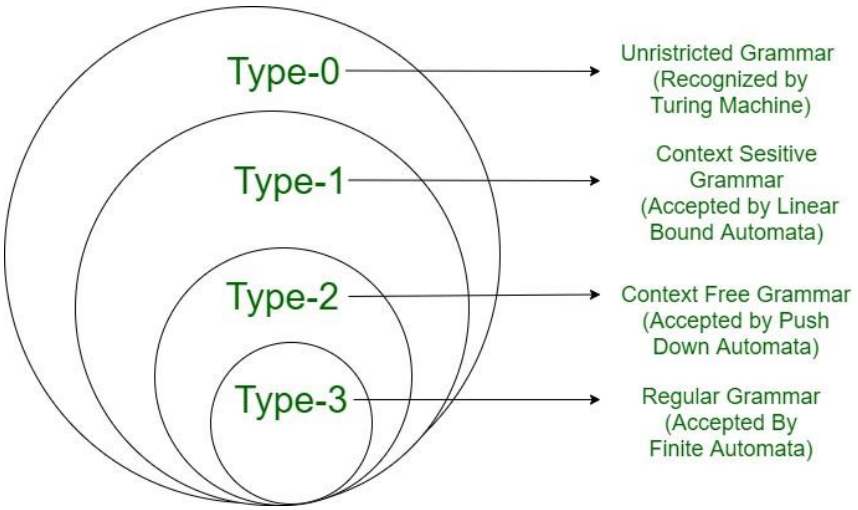
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

I. Dasar Teori

1.1 Automata Theory

Teori Automata merupakan cabang ilmu yang mempelajari logika komputasi yang berkaitan dengan "mesin". Dalam pembahasan teori automata, dikenali automaton yaitu model abstrak dari mesin yang melalui kumpulan "state" yang berubah berdasarkan fungsi transisi yang ditentukan berdasarkan kumpulan masukan simbol, sehingga mesin dapat berubah "state". Kumpulan masukan (input) yang merubah "state" mesin akan menjalankan mesin berdasarkan fungsi transisi dan pada akhirnya menghasilkan suatu kondisi "accept" (diterima) untuk "state" atau kondisi mesin yang telah ditentukan. Berdasarkan lingkup bahasa yang dapat, automaton dibagi beberapa jenis secara hierarkis kemampuan sebuah automaton memeriksa himpunan bahasa / ruang lingkup bahasa yang dapat diperiksa atau dijalankan automaton tersebut, disebut juga *Chomsky Hierarchy*. Berikut merupakan jenis automaton dan tipe bahasa yang dapat diterimanya :

Automaton Family	Type	Grammar (Language)
Finite Automata	Type-3	Regular Grammar
Push Down Automata	Type-2	Context Free Grammar
Linear Bound Automata	Type-1	Context Sensitive Grammar
Turing Machine	Type-0	Unrestricted Grammar



Gambar 1.1 Chomsky Hierarchy

Automaton secara formal didefinisikan terdapat 4 aspek utama yaitu :

1. Input, berupa word yaitu kumpulan alphabet (symbols) yang akan diperiksa automaton dan ditentukan input (word) diterima oleh automaton dengan aturan produksi (production rule) dan state yang ada, yaitu sebuah word terdapat pada language jika diterima automaton (accepting condition).

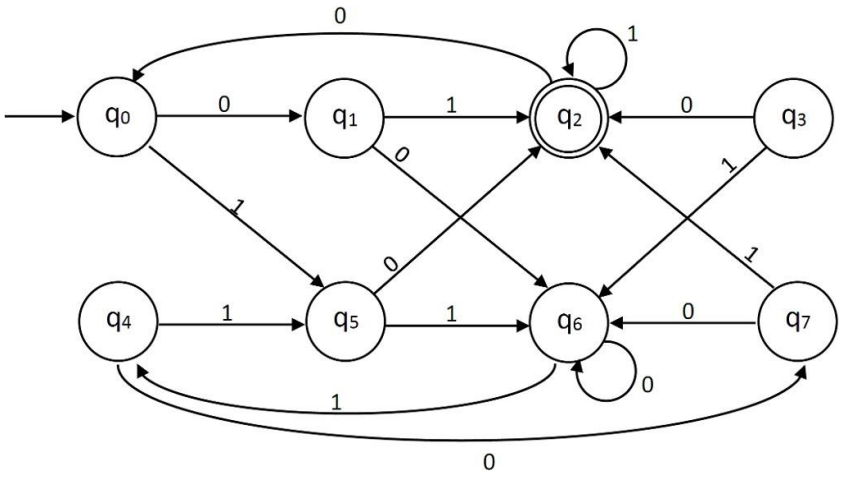
2. States, yaitu kumpulan kondisi (state) automaton, state dapat berupa finite state, stack, tape, dan lainnya berdasarkan definisi dari automaton bersangkutan.
3. Transition function, atau production rule yaitu aturan-aturan yang mendefinisikan berjalannya automaton berdasarkan input symbol dan state dari suatu mesin automaton yang mendefinisikan automaton itu.
4. Acceptance condition, yaitu keadaan ketika automaton menerima sebuah word (input), dapat berupa final state maupun keadaan yang didefinisikan lainnya dari automaton (halt, empty stack, dll).

1.2 Finite Automata (FA)

Finite State Machine atau Finite Automata merupakan mesin yang dapat memeriksa bahasa dalam regular language. Finite Automata didefinisikan dalam bentuk 5 tuple $(Q, \Sigma, \delta, q_0, F)$ yaitu :

1. Q : kumpulan state FA yang finite.
2. Σ : kumpulan symbol input yang finite.
3. δ : transition function, yaitu fungsi yang memetakan dari suatu state pada mesin dan suatu input symbol ke state lain pada mesin FA.
4. q_0 : State awal dari mesin, merupakan state pada Q.
5. F : kumpulan state akhir yang diterima oleh FA.

Secara visualisasi, FA dapat digambarkan sebagai graf berarah dengan setiap simpul merupakan state dari FA (yaitu state pada Q) dan sisi berarah merupakan representasi transtition function FA yang diberikan label simbol input.



Gambar 1.2 Contoh Transition Diagram

FA sendiri dibagi menjadi 2 jenis berdasarkan jenis transition function FA :

1. DFA (Deterministic Finite Automata), yaitu FA dengan transition function dari FA untuk setiap state q pada Q $(q \in Q)$ dan setiap alphabet pada Σ $(\alpha \in \Sigma)$, terdapat tepat satu transition function yang memetakan q ke state lain dengan input symbol α.
2. NFA (Nondeterministic Finite Automata), yaitu FA dengan transition function yang didefinisikan untuk setiap q pada Q $(q \in Q)$ terdapat 0 atau lebih transition function dengan input symbol α pada Σ yang memetakan q ke state lain pada Q.

1.3 Context Free Grammar (CFG)

Context Free Grammar merupakan automata yang termasuk pada type-2 yaitu dapat memeriksa *Context Free Language*. Secara formal CFG didefinisikan sebagai 4 tuple yaitu V, T, P, dan S. Dengan V adalah kumpulan variable / non-terminal dari CFG, T adalah kumpulan terminal dari CFG, P adalah production rule dari CFG, dan S adalah start variable dari CFG.

Context Free Grammar dapat dihasilkan oleh *pushdown automata* seperti halnya *regular languages* dapat dihasilkan oleh *finite state machine*. Karena semua bahasa reguler dapat dihasilkan oleh CFG, semua bahasa reguler juga dapat dihasilkan oleh *pushdown automata*. Bahasa apa pun yang dapat dihasilkan menggunakan *regular expression* dapat dihasilkan oleh *Context Free Grammar*. Cara melakukannya adalah dengan mengambil bahasa reguler, menentukan *finite state machine*, dan menulis aturan produksi yang mengikuti fungsi transisi.

Contoh dari Production Rule CFG :

$S \rightarrow A1B$

$A \rightarrow 0A \mid \epsilon$

$B \rightarrow 0B \mid 1B \mid \epsilon$

1.4 Chomsky Normal Form (CNF)

Sebuah *context-free grammar* dikatakan dalam CNF jika memenuhi *production rule* sebagai berikut:

$A \rightarrow BC$,

$A \rightarrow a$,

$S \rightarrow \varepsilon$

dengan A, B, dan C adalah non-terminal, a adalah terminal, S adalah simbol start, dan ε menandakan string kosong. Selain itu CNF juga harus memenuhi syarat berupa:

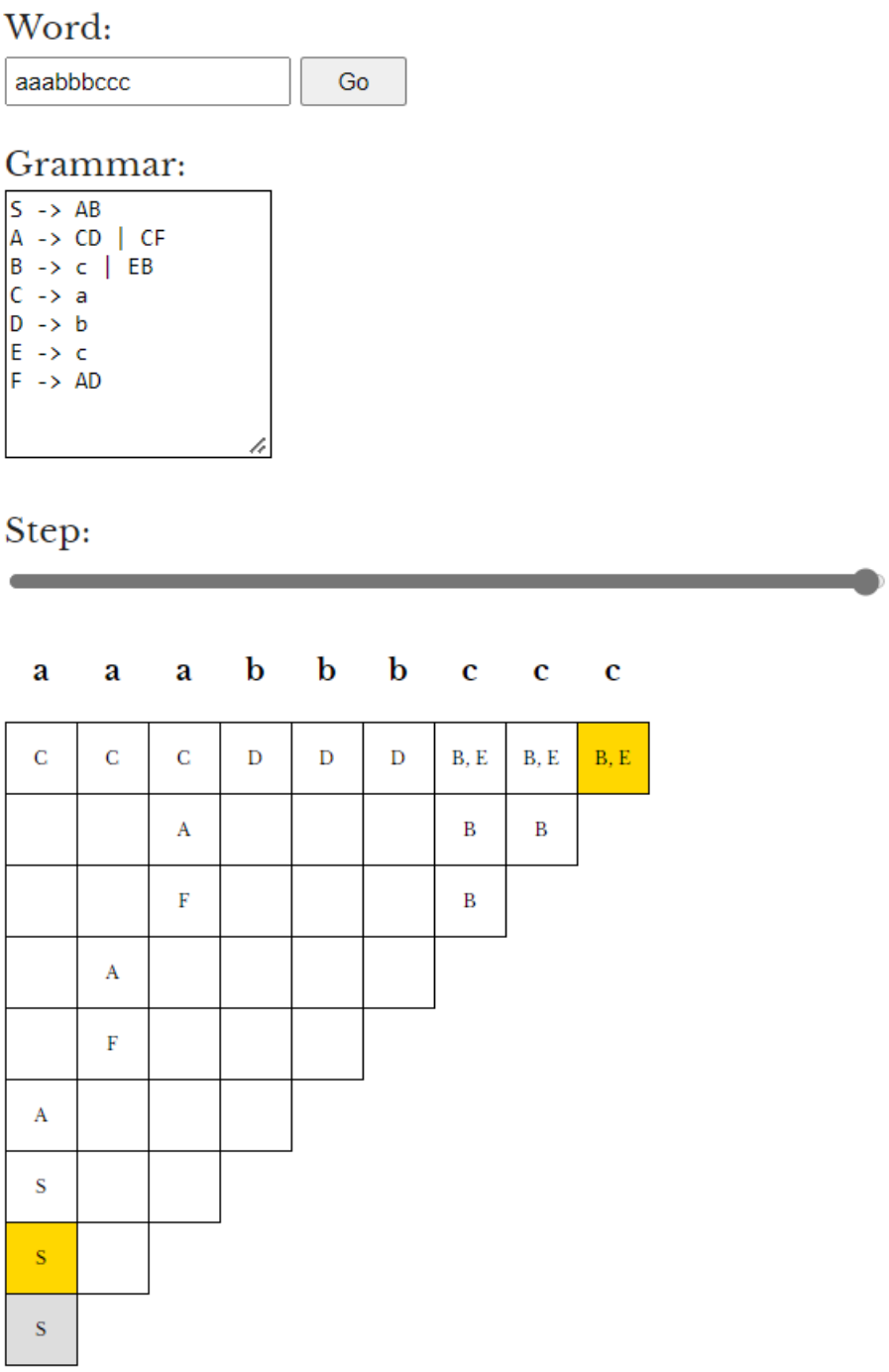
- Tidak memiliki *useless variable*
- Tidak memiliki *ε-production*
- Tidak memiliki *unit production*

Setiap *grammar* dalam CNF adalah CFG dan sebaliknya setiap CFG dapat ditransformasikan ke bentuk yang ekuivalen dengan CNF.

1.5 CYK

Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma penguraian yang sangat efisien untuk *context-free grammar* (CFG). Algoritma ini ideal untuk menentukan masalah kata dengan *context-free grammar* yang telah diubah dalam bentuk *Chomsky normal form* (CNF).

Keunggulan dari algoritma CYK ini adalah efisiensinya yang tinggi. Dengan menggunakan Big O notation, kasus terburuk akan berjalan pada $O(|G| \cdot n^3)$ dengan n adalah panjang dari string yang diuraikan dan |G| adalah ukuran dari CNF.

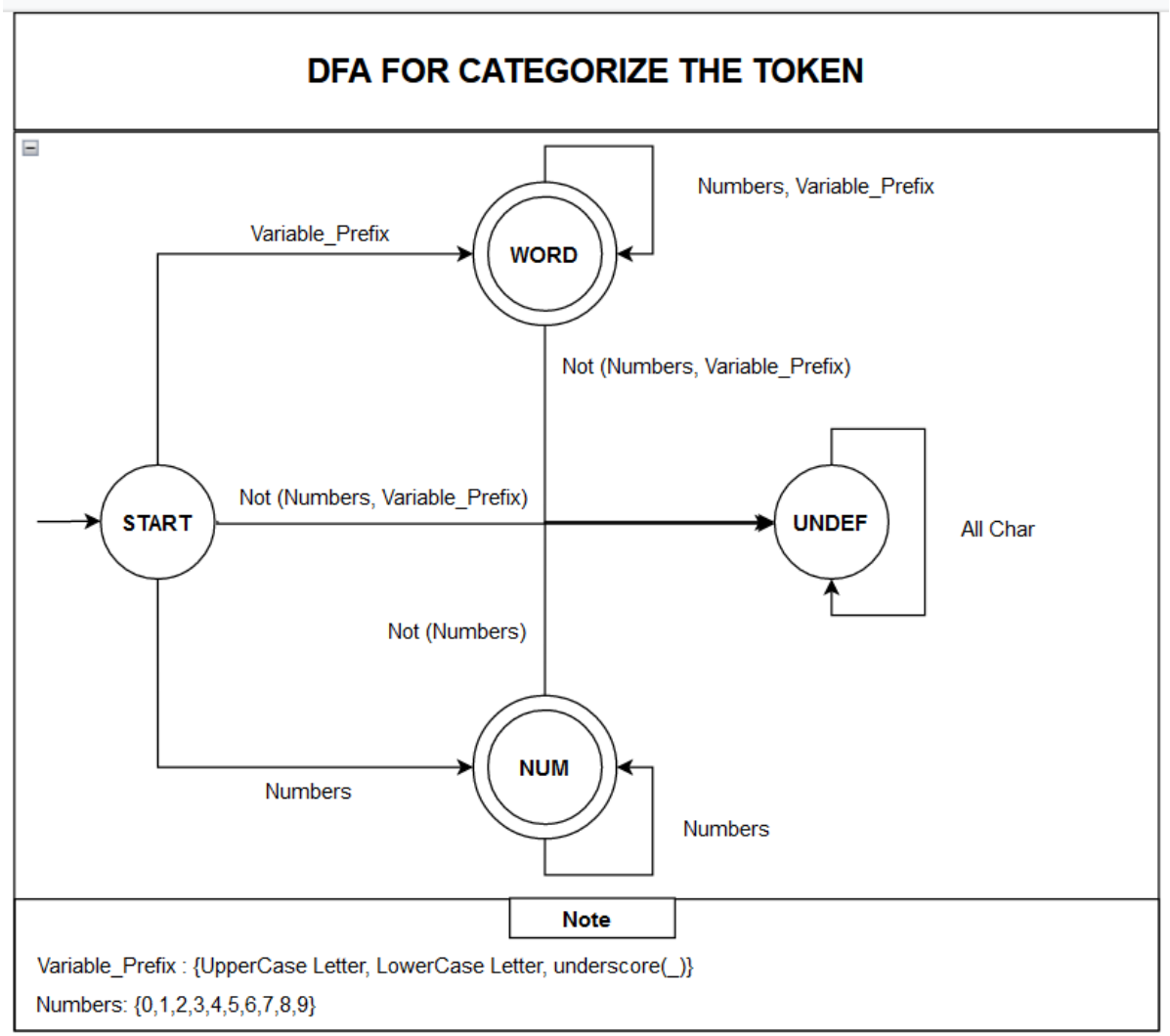


Gambar 1.3 Contoh Word, CNF, dan CYK Table

II. FA dan CFG

2.1 FA

FA yang dibuat merupakan DFA. DFA yang digunakan terdapat satu buah DFA Categorize_Token, berikut merupakan transition diagram DFA.



Gambar 2.1 Gambar DFA Categorize Token

Secara formal DFA Categorize_token didefisikan dengan tuple berisi lima elemen sebagai berikut:

1. DFA = ({start, num, word, undef}, {Numbers, Variable_Prefix}, δ, start, {word, num})

Berikut merupakan penjelasan representasi tiap symbol :

1. Numbers: Kumpulan digit angka {0,1,2,3,4,5,6,7,8,9}
2. Variable_Prefix: Kumpulan alphabet dan underscore (a-z , A-Z, _)
3. All Char: Semua ASCII char

Berikut definisi tiap state pada DFA:

1. start: Start state
2. word: State merepresentasikan token berupa word dan variabel yang valid
3. num: State merepresentasikan token berupa angka yang valid
4. undef: State merepresentasikan token bukan merupakan word, variabel, atau angka yang valid

DFA Categorize_token digunakan untuk memeriksa kebenaran penamaan variabel, word, atau angka.

2.2 CFG

Secara garis besar, berikut adalah CFG yang kami buat.

START -> ALGO

ALGO -> ALGO ALGO | ALGOSC | ALGOSC SEMI_COLON_S | SEMI_COLON_S ALGOSC | SEMI_COLON_S ALGOSC SEMI_COLON_S | IF_STATE | IF_STATE ELSE_STATE | IF_STATE ELSE_IF_STATE | WHILE_CONDITION | FOR_CONDITION | SWITCH_CONDITION | FUNCTION_STATE | TRY_STATE | THROW_STATE | RETURN_STATE

ALGOSC -> ASSIGN | INT_ASSIGN | OBJECT_ASSIGN | METHOD_ALGO_ASSIGN | WORD DOT WORD PARENTHESES_OPEN WORD PARENTHESES_CLOSE | OPERATOR | STRING_OPERATOR | METHOD_ALGO | DELETE_OP | PARAM_STATE

SEMI_COLON_S -> SEMI_COLON | SEMI_COLON SEMI_COLON_S

THROW_STATE -> THROW PARAM_THROW | THROW PARAM_THROW SEMI_COLON_S

PARAM_THROW -> STRING | EXPRES

METHOD_ALGO -> PARAM | WORD DOT WORD PARENTHESES_OPEN METHOD_ALGO PARENTHESES_CLOSE | WORD DOT WORD PARENTHESES_OPEN PARENTHESES_CLOSE | ELMT_LIST

PARAM -> WORD | WORD DOT WORD | WORD DOT METHOD_ALGO | WORD COMMA PARAM | WORD EQUAL STRING | WORD EQUAL STRING COMMA PARAM | WORD EQUAL STRING | WORD EQUAL WORD | STRING | EXPRES | ELMT_LIST | WORD DOT PARAM

PARAM_STATE -> PARENTHESES_OPEN PARAM PARENTHESES_CLOSE | PARENTHESES_OPEN PARENTHESES_CLOSE | PARENTHESES_OPEN PARAM PARENTHESES_CLOSE

SWITCH_CONDITION -> SWITCH PARENTHESES_OPEN PARAM_SWITCH PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE | SWITCH PARENTHESES_OPEN PARAM_SWITCH PARENTHESES_CLOSE CURLY_BRACKET_OPEN CASE_CONDITION CURLY_BRACKET_CLOSE

PARAM_SWITCH -> PARAM_SWITCH COMMA PARAM_SWITCH | EXPRES | STRING | ELMT_LIST

CASE_CONDITION -> CASE_CONDITION CASE_CONDITION | CASE_ALGO DEFAULT_CONDITION | DEFAULT_CONDITION CASE_ALGO | CASE_ALGO | DEFAULT_CONDITION

CASE_ALGO -> CASE_ALGO CASE_ALGO | CASE PARENTHESES_OPEN PARAM_SWITCH PARENTHESES_CLOSE COLON SWITCH_CASE_ALGO | CASE PARENTHESES_OPEN PARAM_SWITCH PARENTHESES_CLOSE COLON

SWITCH_CASE_ALGO -> SWITCH_CASE_ALGO SWITCH_CASE_ALGO | ALGO | SWITCH_CONDITION

DEFAULT_CONDITION -> DEFAULT COLON SWITCH_CASE_ALGO | DEFAULT COLON

IF_STATE -> IF_ALGO | IF_ALGO ELSE_STATE | IF_ALGO ELSE_IF_STATE

IF_ALGO -> IF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | IF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE ALGO | IF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

ELSE_IF_STATE -> ELSE_IF_ALGO | ELSE_IF_ALGO ELSE_STATE | ELSE_IF_ALGO ELSE_IF_STATE

ELSE_IF_ALGO -> ELIF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | ELIF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE ALGO | ELIF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

ELSE_STATE -> ELSE CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | ELSE ALGO | ELSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

CONDITION -> PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE | CONDITION LOGIC_OP CONDITION | EXPRES RELAT_OP EXPRES | NOT_BITWISE CONDITION | WORD | BOOLEAN

WHILE_CONDITION -> WHILE PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO_LOOP CURLY_BRACKET_CLOSE | WHILE PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

FOR_CONDITION -> FOR PARENTHESES_OPEN FOR_TYPE_CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO_LOOP CURLY_BRACKET_CLOSE | FOR PARENTHESES_OPEN FOR_TYPE_CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

FOR_TYPE_CONDITION -> FOR_A | FOR_B

FOR_A -> WORD IN WORD | LVC WORD IN WORD

FOR_B -> COND_1 SEMI_COLON COND_2 SEMI_COLON COND_3 | SEMI_COLON COND_2 SEMI_COLON COND_3

COND_1 -> LET WORD EQUAL INT | LET WORD EQUAL INT COMMA ADDITION_EXPRES | WORD EQUAL INT | WORD EQUAL INT COMMA ADDITION_EXPRES

COND_2 -> WORD RELAT_OP METHOD_ALGO

COND_3 -> WORD INCREMENT | WORD DECREMENT | INT_ASSIGN

ADDITION_EXPRES -> ASSIGN | ADDITION_EXPRES COMMA ADDITION_EXPRES

ALGO_LOOP -> CONTINUE_STATE | BREAK_STATE | BREAK | ALGO_LOOP ALGO_LOOP | ALGO_LOOPSC | ALGO_LOOPSC SEMI_COLON_S | SEMI_COLON_S ALGO_LOOPSC | SEMI_COLON_S ALGO_LOOPSC SEMI_COLON_S | IF_LOOP_STATE | IF_LOOP_STATE ELSE_IF_LOOP_STATE | IF_LOOP_STATE ELSE_LOOP_STATE | ALGO

ALGO_LOOPSC -> CONTINUE_STATE | BREAK_STATE | BREAK

IF_LOOP_STATE -> IF_LOOP_ALGO | IF_LOOP_ALGO ELSE_LOOP_STATE | IF_LOOP_ALGO ELSE_IF_LOOP_STATE

IF_LOOP_ALGO -> IF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO_LOOP CURLY_BRACKET_CLOSE | IF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE ALGO_LOOP

ELSE_IF_LOOP_STATE -> ELSE_IF_ALGO | ELSE_IF_ALGO ELSE_LOOP_STATE | ELSE_IF_LOOP_ALGO ELSE_IF_LOOP_STATE

ELSE_IF_LOOP_ALGO -> ELIF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO_LOOP CURLY_BRACKET_CLOSE | ELIF PARENTHESES_OPEN CONDITION PARENTHESES_CLOSE ALGO_LOOP

ELSE_LOOP_STATE -> ELSE CURLY_BRACKET_OPEN ALGO_LOOP CURLY_BRACKET_CLOSE | ELSE CURLY_BRACKET_OPEN IF_LOOP_STATE CURLY_BRACKET_CLOSE | ELSE ALGO_LOOP | ELSE IF_LOOP_STATE

LIST -> SQUARE_BRACKET_OPEN ELMT_LIST SQUARE_BRACKET_CLOSE | SQUARE_BRACKET_OPEN SQUARE_BRACKET_CLOSE | PARENTHESES_OPEN PARENTHESES_CLOSE | PARENTHESES_OPEN ELMT_LIST PARENTHESES_CLOSE

ELMT_LIST -> LIST | LIST COMMA ELMT_LIST | EXPRES | EXPRES COMMA ELMT_LIST | STRING | STRING COMMA ELMT_LIST

EXPRES -> METHOD_ALGO PARENTHESES_OPEN EXPRES PARENTHESES_CLOSE | PARENTHESES_OPEN EXPRES PARENTHESES_CLOSE | NOT_BITWISE EXPRES | STRING | WORD | INT | FLOAT | NEG_FL | NEG_INT | PARENTHESES_OPEN EXPRES PARENTHESES_CLOSE | EXPRES ARITH_OP EXPRES | WORD DOT WORD | NULL

DELETE_OP -> DELETE METHOD_ALGO

TRY_STATE -> TRY CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE CATCH_STATE | TRY CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE CATCH_STATE | TRY CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE FINAL_STATE | TRY CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE FINAL_STATE | TRY CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE CATCH_STATE FINAL_STATE | TRY CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE CATCH_STATE FINAL_STATE

CATCH_STATE -> CATCH PARENTHESES_OPEN WORD PARENTHESES_CLOSE CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | CATCH PARENTHESES_OPEN WORD PARENTHESES_CLOSE CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE | CATCH CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | CATCH CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

FINAL_STATE -> FINALLY CURLY_BRACKET_OPEN ALGO CURLY_BRACKET_CLOSE | FINALLY CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

BREAK_STATE -> BREAK WORD

CONTINUE_STATE -> CONTINUE WORD | CONTINUE

FUNCTION_STATE -> FUNCTION VAR_FUNCTION | FUNCTION VAR_FUNCTION CURLY_BRACKET_OPEN ALGO_IN_FUNCTION CURLY_BRACKET_CLOSE | FUNCTION VAR_FUNCTION CURLY_BRACKET_OPEN FUNCTION_STATE CURLY_BRACKET_CLOSE | FUNCTION VAR_FUNCTION CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE

VAR_FUNCTION -> WORD PARENTHESES_OPEN PARENTHESES_CLOSE | WORD

PARENTHESES_OPEN PARAM_FUNCTION PARENTHESES_CLOSE

PARAM_FUNCTION -> WORD | WORD COMMA PARAM_FUNCTION

ALGO_IN_FUNCTION -> RETURN_STATE ALGO_IN_FUNCTION | RETURN_STATE SEMI_COLON_S ALGO_IN_FUNCTION | RETURN_STATE SEMI_COLON_S | RETURN_STATE | ALGO | ALGO ALGO_IN_FUNCTION | ALGO_IN_FUNCTION ALGO

RETURN_STATE -> RETURN | RETURN RETURN_PARAM | RETURN SEMI_COLON_S | RETURN RETURN_PARAM SEMI_COLON_S

RETURN_PARAM -> RETURN_OBJ | RETURN_OBJ COMMA RETURN_PARAM

RETURN_OBJ -> BOOLEAN | STRING | WORD | LIST | INT | FLOAT | NEG_FL | NEG_INT | PARENTHESES_OPEN EXPRES PARENTHESES_CLOSE | WORD DOT WORD | NULL | METHOD_ALGO

OBJECT_ASSIGN -> LVC WORD EQUAL OBJECT_LIST | WORD EQUAL OBJECT_LIST

OBJECT_LIST -> CURLY_BRACKET_OPEN CURLY_BRACKET_CLOSE | CURLY_BRACKET_OPEN OBJECT_ELMT CURLY_BRACKET_CLOSE

OBJECT_ELMT -> WORD COLON OBJECT_VALUE | WORD COLON OBJECT_VALUE COMMA OBJECT_ELMT

OBJECT_VALUE -> EXPRES | STRING | WORD

OPERATOR -> WORD EQUAL EXPRES | PARENTHESES_OPEN WORD PARENTHESES_CLOSE

ASSIGN -> ASSIGN_LVC | LVC ASSIGN_LVC

ASSIGN_LVC -> WORD EQUAL ASSIGN_LVC | WORD EQUAL EXPRES | WORD EQUAL STRING | WORD EQUAL LIST

INT_ASSIGN -> LVC WORD OP_ASSIGN EXPRES | WORD OP_ASSIGN EXPRES

METHOD_ALGO_ASSIGN -> METHOD_ALGO EQUAL EXPRES | METHOD_ALGO EQUAL STRING | METHOD_ALGO EQUAL LIST | METHOD_ALGO EQUAL EXPRES | METHOD_ALGO EQUAL STRING | METHOD_ALGO EQUAL LIST

LVC -> LET | VAR_EXPRES | CONST

OP_ASSIGN -> EQUAL | PLUS EQUAL | MINUS EQUAL | TIMES EQUAL | DIVISION EQUAL | BITWISE_OPERATOR EQUAL

COMMENT_OP -> COMMENT_ALONE SENTENCE | COMMENT_OPEN COMMENT_LONG COMMENT_CLOSE

COMMENT_LONG -> SENTENCE | SENTENCE COMMENT_LONG

SENTENCE -> WORD | WORD SENTENCE

RELAT_OP -> GREATER | LESS | GREATER_AND_EQUAL | LESS_AND_EQUAL | EQUAL_TO | NOT_EQUAL_TO | EQUAL_VALUE_AND_EQUAL_TYPE | NOT_EQUAL_VALUE_OR_TYPE

ARITH_OP -> MINUS | PLUS | TIMES | DIVISION | MODULO | EXPONENT |

BITWISE_OPERATOR

LOGIC_OP -> AND | OR

BITWISE_OPERATOR -> F_RIGHT_SHIFT | RIGHT_SHIFT | LEFT_SHIFT | AND_BITWISE | XOR_BITWISE | OR_BITWISE

STRING -> QUOTE W_STRING QUOTE | DOUBLE_QUOTE W_STRING DOUBLE_QUOTE

W_STRING -> WORD | WORD W_STRING

PARENTHESES_WORD -> QUOTE WORD QUOTE | DOUBLE_QUOTE WORD DOUBLE_QUOTE

DLL

III. Implementasi

3.1 Struktur Data

Empat file utama yang menyusun program kami adalah main.py, tokenizer.py, cfg_to_cnf.py, dan cyk_op.py. main.py berfungsi untuk menjalankan program pengecekan syntax file *.js. tokenizer.py berfungsi untuk mem-*parse* file javascript menjadi token-token yang digabung dalam sebuah list. tokenizer.py juga berfungsi untuk mengecek kategori dari suatu token nonterminal ("word" jika merupakan sebuah string atau nama variabel, "num" jika merupakan sebuah angka, dan "undef" jika bukan merupakan keduanya [Salah satu penanda syntax error pada file *.js]). setelah file input di-*parse*, token-token tersebut akan di cek kebenarannya dengan menggunakan algoritma CYK (Cocke-Younger-Kasami). algoritma tersebut akan mengecek apakah token-token yang dimasukan sesuai dengan grammar (CNF) yang telah dibuat sebelum dilakukan check validity. Jika file *.js tersebut bisa dicapai pada grammar, program akan menampilkan "*Accepted"*. Namun, file *.js tidak bisa dicapai pada grammar yang telah dibuat, program akan menampilkan "*Syntax Error*".

3.2 Fungsi dan Prosedur

3.2.1 main.py

Pada main.py tidak terdapat fungsi atau prosedur, file ini hanya memanggil fungsi atau prosedur dari file lain.

3.2.2 cyk_op.py

Pada cyk_op.py terdapat dua fungsi atau prosedur, yaitu:

- cyk_algorithm

  Parameter:
  file_terminal, cnf_grammar, file_input.

  Proses:
  Melakukan CYK algorithm terhadap file_input dan membuat cyk_table yang berisi state-state.

  Return:
  Elemen puncak dari cyk_table (elemen terakhir).

- check_validity
  Parameter:
  file_terminal, cnf_grammar, file_input.

  Initial State:
  semua parameter terdefinisi, file_terminal berisi string nama file terminal, cnf_grammar berisi dictionary cnf, dan file_input berisi nama file yang akan diperiksa.

  Final State:
  Menampilkan “Accepted” jika file yang dibaca valid secara syntax dan menampilkan “Syntax Error” jika file yang dibaca tidak valid secara syntax.

3.2.3 cfg_to_cnf.py

Pada cfg_to_cnf.py terdapat tujuh fungsi atau prosedur, yaitu:

- simplify_cfg
  Parameter:
  cfg_grammar.

  Initial State:
  semua parameter terdefinisi, cfg_grammar adalah dictionary cfg yang dibaca dari file “cfg.txt”.

  Final State:
  cfg_grammar telah disimplifikasi (menghapus unit production dan useless symbol).

- cnf_algorithm
  Parameter:
  cfg_grammar.

  Initial State:
  semua parameter terdefinisi, cfg_grammar adalah dictionary cfg yang telah disimplifikasi.

  Final State:
  cfg_grammar telah menjadi cnf_grammar (Production rule hanya terdiri atas maksimal dua variabel atau satu terminal).

- write_cnf_file
  Parameter:
  cnf_grammar.

  Initial State:
  semua parameter terdefinisi, cnf_grammar adalah dictionary cnf.

  Final State:
  File “cnf.txt” tertulis di dalam folder lib yang berisi representasi dari cnf_grammar.

- read_grammar_text
  Parameter:
  grammar_text.

  Proses:
  Membaca representasi txt dari grammar (cfg atau cnf), lalu mengubahnya ke dalam bentuk dictionary.

  Return:
  Dictionary dari grammar.

- read_input
  Parameter:
  file_input_name.

  Proses:
  Membaca representasi file input(*.js), lalu mengubahnya ke dalam list of token (list of string).

  Return:
  List of tokens dari input.

- read_terminal
  Parameter:
  terminal_file_name.

  Proses:
  Membaca file terminal.txt dan file terminal_rule.txt, lalu mengubahnya ke dalam bentuk list of token (list of string).

  Return:
  tuple berisi list of string terminal, dan list of string terminal_rule.

- convert_cfg
  Parameter:
  cfg_text.

  Initial State:
  semua parameter terdefinisi, cfg_text adalah nama file (“cfg.txt”) yang merupakan representasi cfg dalam txt.

  Final State:
  File “cnf.txt” tertulis di dalam folder lib yang berisi representasi dari cnf_grammar.

3.2.4 tokenizer.py

Pada cfg_to_cnf.py terdapat dua fungsi atau prosedur, yaitu:

- categorize_token
  Parameter:
  token.

Proses:
menentukan kategori dari sebuah token nonterminal.

Return:
Kategori token (“word”, “num”, atau “undef”).

- js_to_token
  Parameter:
  Js_file.

  Proses:
  mengubah representasi file *.js yang akan diperiksa ke dalam bentuk list of token (string).

  Return:
  list of token.

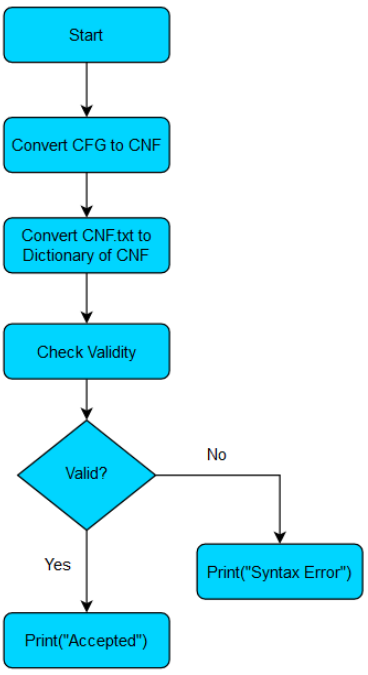
3.3 Antarmuka

Pada tugas besar ini digunakan Antarmuka yang berbasis CLI (*Command Line Interface*). Program dapat dijalankan pada terminal/*command prompt* dengan working directory src(lokasi main.py berada), setelah itu panggil dengan format “python main.py nama_file.js” tanpa tanda *double quote (“)*. File yang akan diperiksa harus dipastikan berada di dalam folder lib. Setelah itu akan keluar hasil apakah terdapat *syntax error* pada program atau *accepted.*

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py a.js
Syntax Error
```

Gambar 3.1 Tampilan program pada cmd

3.4 Flow Program



Gambar 3.1 Flowchart Program

IV. Pengujian / Program Testing

4.1 tesAssignment.js

src > lib > test > tesAssignment.js > ...

```
1   let angka = 1
2   let kata = 'aa'
3   const listAngka = [1, 2, 3]
4   nama = "luffy"
5   problem = null
6   let d =2
7   let x=y=z=f=g=3
8   return x
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesAssignment.js
Accepted
```

Source code pada tesAssignment.js accepted karena assignmentnya telah sesuai ketentuan yaitu nama variabel valid(LHS) dan RHS juga valid.

src > lib > test > tesAssignment.js > ...

```
1   let angka == 1
2   let kata = 'aa'
3   const listAngka = [1, 2, 3]
4   nama = "luffy"
5   problem = null
6   let d =2
7   let x=y=z=f=g=3
8   return x
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesAssignment.js
Syntax Error
```

Source code pada tesAssignment.js syntax error karena assignment variable tidak boleh menggunakan operator equal to.

4.2 tesComment.js

GaruParsing > src > lib > test > tesComment.js

```
1   // KAIZOKU OU NI ORE WA NARU
2   // ORE NA WA EREN JAEGER
3   // DAGA KOTOWARU
4   /*
5       SOUKA NA, MURI DESU YO
6       SOUKA NA, MURI DESU YO
7       SOUKA NA, MURI DESU YO
8   */
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesComment.js
Accepted
```

Source code pada tesComment.js accepted karena hanya terdiri dari comment single line dan comment multiline yang tertutup.

src > lib > test > tesComment.js

```
1  / KAIZOKU OU NI ORE WA NARU
2  // ORE NA WA EREN JAEGER
3  // DAGA KOTOWARU
4  /*
5      SOUKA NA, MURI DESU YO
6      SOUKA NA, MURI DESU YO
7      SOUKA NA, MURI DESU YO
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesComment.js
Syntax Error
```

Source code pada tesComment.js syntax error karena comment single line hanya terdiri dari satu slash dan comment multi line tidak ditutup.

4.3 tesIf.js

GaruParsing > src > lib > test > tesIf.js

```
1    a = 2
2    if (a == 2) {
3        rob2(tree)
4    }
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesIf.js
Accepted
```

Source code pada tesIf.js accepted karena sesuai dengan ketentuan syntax, yaitu if (condition) {procedure}.

src > lib > test > tesIf.js

```
1  1a = 2
2  if (a == 2) {
3      rob2(tree)
4  }
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesIf.js
Syntax Error
Error pada line ke- 1
```

Source code pada tesIf.js syntax error karena variable tidak bisa didahului dengan angka.

4.4 tesIfElse.js

GaruParsing > src > lib > test > tesIfElse.js

```
1    a = 2
2    b = 3
3
4    if (a >= b) {
5        max = a;
6    } else {
7        max = b;
8    }
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesIfElse.js
Accepted
```

Source code pada tesIfElse.js accepted karena sesuai dengan ketentuan syntax, yaitu if (condition) { procedure }else{ procedure}.

src > lib > test > tesIfElse.js

```
1   a = 2
2   b = 3
3
4   else {
5       max = b;
6   }
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesIfElse.js
Syntax Error
```

Source code pada tesIfElse.js syntax error karena else tidak didahului oleh if.

4.5 tesIfElseIf.js

GaruParsing > src > lib > test > tesIfElseIf.js

```
1   a = 2
2   b = 3
3
4   if (a > b) {
5       console.log("a > b");
6   } else if (b > a) {
7       console.log("a < b");
8   }
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesIfElseIf.js
Accepted
```

Source code pada tesIfElseIf.js accepted karena sesuai dengan ketentuan syntax, yaitu if (condition) { procedure }else if(condition){ procedure}.

src > lib > test > tesIfElseIf.js

```
1   a = 2
2   b = 3
3
4   else if (b > a) {
5       console.log("a < b");
6   }
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesIfElseIf.js
Syntax Error
```

Source code pada tesIfElseIf.js syntax error karena else if tidak didahului oleh if.

4.6 tesTryCatchFinally.js

GaruParsing > src > lib > test > tesTryCatchFinally.js

```
1    try { console.log(a) }
2    catch { a = b }
3    finally { kata = "mugiwara" }
4
5    try {}
6    catch {}
7    finally { A = 1 }
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesTryCatchFinally.js
Accepted
```

Source code pada tesTryCatchFinally.js accepted karena sesuai dengan ketentuan syntax, yaitu try{procedure} catch{procedure} finally{procedure}.

src > lib > test > tesTryCatchFinally.js

```
1  { console.log(a) } catch { a = b } finally { kata = "mugiwara" }
2  try {} catch {} finally { A = 1 }
```

```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesTryCatchFinally.js
Syntax Error
```

Source code pada testTryCatchFinally.js syntax error karena tidak didahului oleh try.

4.7 tesThrowDelete.js

GaruParsing > src > lib > test > tesThrowDelete.js > ...

```
...
1    a = 5
2    b = 7
3    delete a
4    A = (1,2,3)
5    let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
6    throw person.eyeColor
7    throw "ASASA"
```

```
PS D:\KULIAH\SEMESTER 3\Teori Bahasa Formal dan Otomata\Tugas Besar\GaruParsing\src> python main.py tesThrowDelete.js
Accepted
```

Source code pada tesThrowDelete.js accepted karena sesuai dengan ketentuan syntax, yaitu throw object.

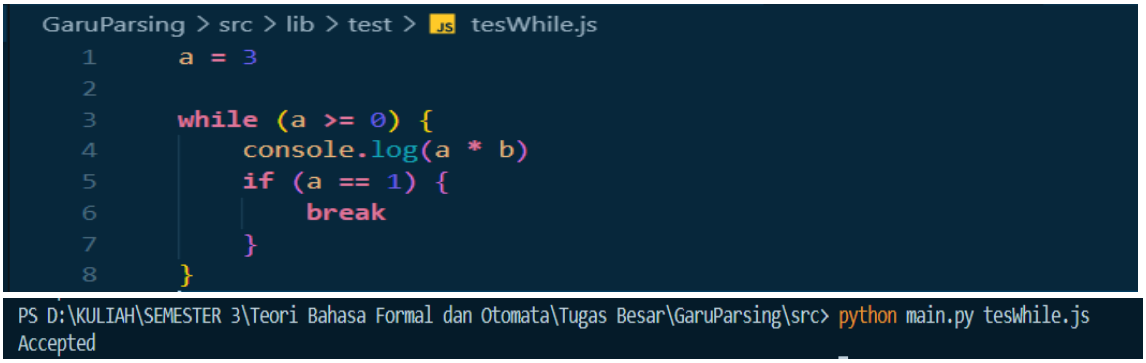
src > lib > test > tesThrowDelete.js > ...

```
1  a = 5
2  b = 7
3  delete
4  A = (1,2,3)
5  let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
6  throw person.eyeColor
7  throw "ASASA"
```

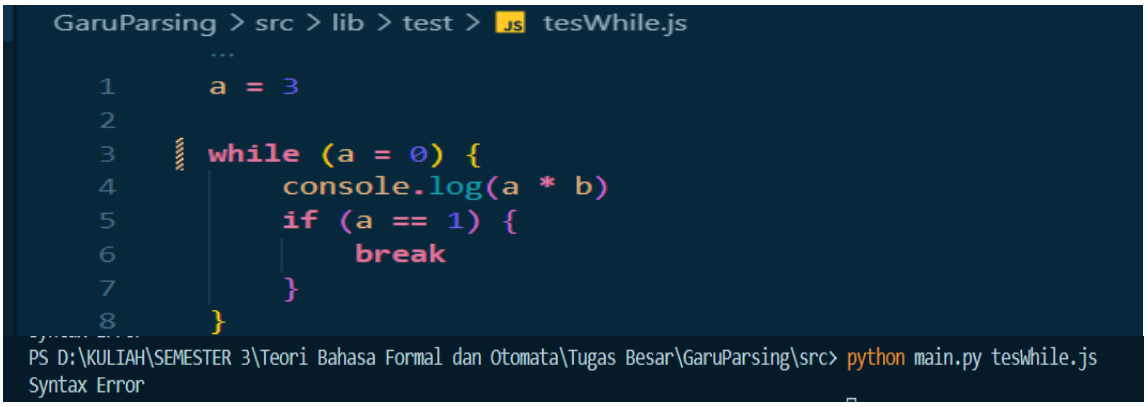
```
PS C:\Kuliah\Jurusan - IF\IF-Semester3\TBFO\Tubes\GaruParsing\src> py main.py tesThrowDelete.js
Syntax Error
```

Source code pada tesThrowDelete.js syntax error karena delete tidak diisi dengan paramater / variable yang harus di delete.

4.8 tesWhile.js

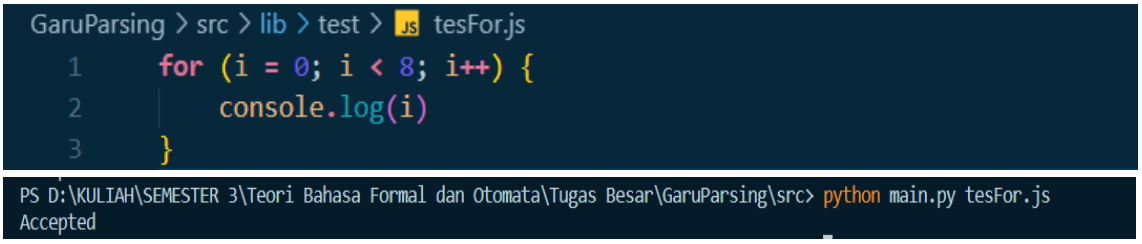


Source code pada tesWhile.js accepted karena sesuai dengan ketentuan syntax, yaitu while (condition) { procedure }.



Source code pada tesWhile.js syntax error karena pada bagian condition terdapat assignment yang seharusnya berisikan boolean.

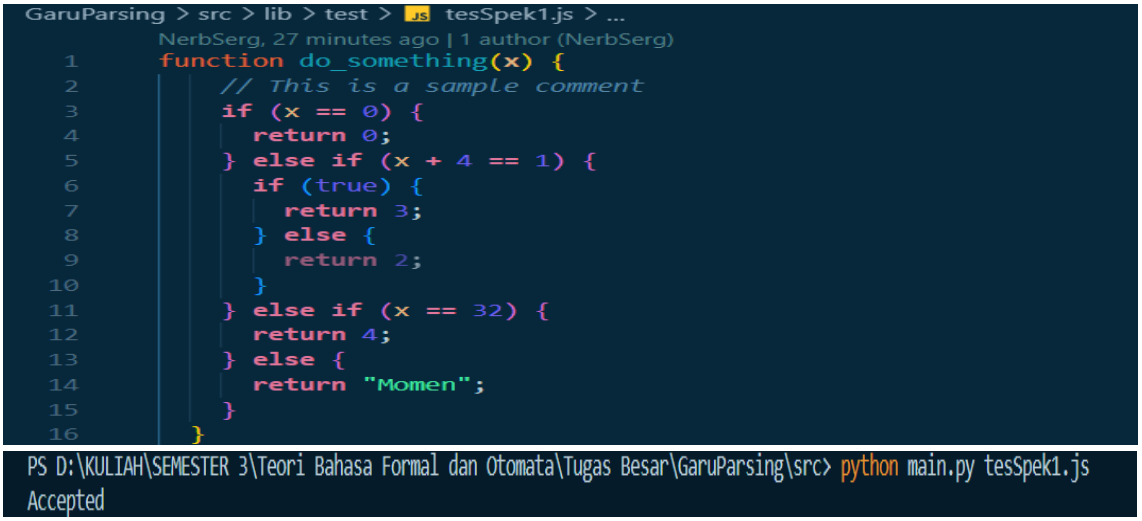
4.9 tesFor.js



Source code pada tesFor.js accepted karena sesuai dengan ketentuan syntax, yaitu for (start; condition; final) { procedure }.

Source code pada tesFor.js syntax error karena pada argumen di dalam for, tidak terdapat semicolon (;) antara bagian condition dengan final.

4.10 tesSpek1.js



Source code pada tesSpek1.js accepted karena sesuai dengan ketentuan syntax.

4.11 tesSpek2.js

Source code tesSpek2.js mengalami syntax error karena pada else if , bagian condition tidak berada di dalam tanda kurung (parentheses).

V. Link Repository

Berikut link repository kami:

https://github.com/Mehmed13/GaruParsing.git

VI. Pembagian Tugas

No.	NIM	Nama	Tugas
1.	13521043	Nigel Sahl	CFG
2.	13521058	Ghazi Akmal Fauzan	Converter CFG to CNF
3.	13521066	Muhammad Fadhil Amri	Tokenizer dan CYK

VII. Daftar Referensi

Basics of Automata Theory - Stanford
The CYK Algorithm - Xarg
The CYK Algorithm - Wikipedia
CYK Algorithm for Context Free Grammar - GeeksforGeeks
Parsing in Python: all the tools and libraries you can use - Tomassetti
Automata Chomsky's Normal Form (CNF) - Javatpoint
Chomsky Normal Form & CFG to CNF Conversion - YouTube
Context Free Grammars - Brilliant