

Laporan Tugas Besar

Parser Bahasa JavaScript (Node.js)

Mata Kuliah IF2124 - Teori Bahasa Formal dan Otomata



Kelompok LSN - Kelas 02

Nama Anggota:

Michael Leon Putra Widhi 13521108

Satria Octavianus Nababan 13521168

Nathan Tenka 13521172

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022/2023

Daftar Isi

BAB 1 – Teori Dasar	3
A. Finite Automata	3
B. Context-Free Grammar	4
C. Chomsky Normal Form	4
BAB 2 - Hasil Finite Automata dan Context-Free Grammar	7
Context-Free Grammar	7
Finite Automata	14
BAB 3 - Implementasi dan Pengujian	18
A. Implementasi pada <i>Source Code</i>	18
B. Pengujian	21
BAB 4 - Penutup	33
A. Kesimpulan	33
B. Saran	33
C. Lampiran	34
1. Link Repository Github	34
2. Pembagian Tugas	34

BAB 1

Teori Dasar

A. Finite Automata

Finite Automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata di mana sistem dapat berada di salah satu dari sejumlah berhingga konfigurasi internal disebut state. Beberapa contoh sistem dengan state berhingga antara lain pada mesin minuman otomatis atau *vending machine*, pengatur lampu lalu lintas dan *lexical analyser*.

Suatu finite automata terdiri dari beberapa bagian. Finite automata mempunyai sekumpulan state dan aturan-aturan untuk berpindah dari state yang satu ke state yang lain, tergantung dari simbol nya. Finite automata mempunyai state awal, sekumpulan state dan state akhir. Finite automata merupakan kumpulan dari lima elemen atau dalam bahasa matematis dapat disebut sebagai 5-tuple. Definisi formal dari finite automata dikatakan bahwa finite automata merupakan list dari 5 komponen : kumpulan state, input, aturan perpindahan, state awal, dan state akhir.

Dalam DFA sering digunakan istilah fungsi transisi untuk mendefinisikan aturan perpindahan, biasanya dinotasikan dengan δ . Jika finite automata memiliki sebuah panah dari suatu state x ke suatu state y , dan memiliki label dengan simbol input 0 , ini berarti bahwa, jika automata berada pada state x ketika automata tersebut membaca 0 , maka automata tersebut dapat berpindah ke state y dapat diindikasikan hal yang sama dengan fungsi transisi dengan mengatakan bahwa $\delta(x, 0) = y$.

Sebuah finite automata terdiri dari lima komponen $(Q, \Sigma, \delta, q_0, F)$, di mana :

- Q adalah himpunan set berhingga yang disebut dengan himpunan states.
- Σ adalah himpunan berhingga alfabet dari simbol .

c. $\delta : Q \times \Sigma$ adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state. Fungsi transisi sering dilambangkan dengan δ .

d. $q_0 \in Q$ adalah states awal.

e. $F \subseteq Q$ adalah himpunan states akhir.

B. Context-Free Grammar

Dalam teori bahasa formal, Context-Free Grammar (CFG) adalah sebuah tata bahasa formal dengan bentuk

$$A \rightarrow \alpha$$

Dengan A adalah sebuah symbol non-terminal, dan α adalah terminal dan atau nonterminal. Context-Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti tata bahasa regular yaitu menunjukkan bagaimana menghasilkan suatu bagian-bagian (untai) dalam sebuah bahasa. Context-Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree / parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan. Contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

Dengan penurunan sebagai berikut, akan menghasilkan aabbaa :

$$S \rightarrow aSa \rightarrow aaSaa \rightarrow aabSbaa \rightarrow aabbaa$$

C. Chomsky Normal Form

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk Context-Free Grammar (CFG) . Bentuk normal Chomsky dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit, dan ε . Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi bentuk normal Chomsky dengan syarat tata bahasa bebas konteks tersebut:

- a. Tidak memiliki produksi useless
- b. Tidak memiliki produksi unit
- c. Tidak memiliki produksi ϵ

Bentuk normal Chomsky (Chomsky Normal Form, CNF) adalah Context-Free Grammar (CFG) dengan setiap produksinya berbentuk :

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

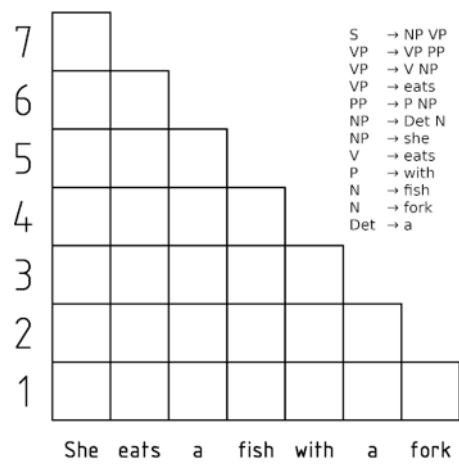
D. Cocke-Younger-Kasami (CYK)

Algoritma Cocke-Younger-Kasami (CYK) adalah algoritma untuk menentukan apakah suatu untai (string) dapat diterima oleh suatu Bahasa-BebasKonteks (Context Free Grammar – CFG). CFG yang diterima oleh algoritma CYK yang diterangkan dalam makalah ini adalah CFG dalam bentuk norma Chomsky Normal Form (CNF). Apabila suatu untai dapat diterima, dipaparkan tahapan algoritma CYK untuk menyelesaikan permasalahan ini. Algoritma yang akan dibahas merupakan metode yang umum digunakan dalam teori otomata dan bahasa formal terutama di bidang desain kompiler bahasa pemrograman. Proses parsing untai dengan algoritma CYK memanfaatkan struktur data sebuah array dua dimensi dan merupakan aplikasi Pemrograman Dinamis (Program Dinamis) karena proses parsing memanfaatkan hasil parsing sebelumnya untuk memutuskan apakah proses yang sedang berlangsung dapat diterima maupun tidak. Dalam makalah ini akan dibahas sebuah contoh pembentukan array dengan CYK yang merupakan algoritma terdiri dari banyak iterasi kolom-baris, kemudian digambarkan pohon parsing

Versi standar CYK hanya beroperasi pada tata bahasa bebas konteks yang diberikan dalam bentuk normal Chomsky (CNF). Namun tata bahasa bebas konteks apa pun dapat diubah (setelah konvensi) menjadi tata bahasa CNF yang mengekspresikan bahasa yang sama (Sipser 1997) Pentingnya algoritma CYK berasal dari efisiensi tinggi dalam situasi tertentu. Ini menjadikannya salah satu algoritme penguraian paling efisien dalam hal kompleksitas asimtotik kasus terburuk, meskipun lain ada dengan waktu berjalan rata-rata yang lebih baik dalam banyak skenario praktis.

Algoritma pemrograman dinamis membutuhkan tata bahasa bebas konteks untuk dirender ke dalam bentuk normal Chomsky (CNF), karena menguji kemungkinan untuk

membagi urutan saat ini menjadi dua urutan yang lebih kecil. Tata bahasa bebas konteks apa pun yang tidak menghasilkan string kosong dapat direpresentasikan dalam CNF hanya menggunakan aturan produksi dari bentuk $A \rightarrow BC$. Dalam istilah informal, algoritme ini menganggap setiap kemungkinan substring dari string input dan disetel menjadi true jika substring dengan panjang mulai dari dapat dibangkitkan dari nonterminal. Setelah mempertimbangkan substring dengan panjang 1, ia melanjutkan ke substring dengan panjang 2, dan seterusnya. Untuk substring dengan panjang 2 dan lebih besar, ia mempertimbangkan setiap kemungkinan partisi substring menjadi dua bagian, dan memeriksa untuk melihat apakah ada beberapa produksi yang cocok dengan bagian pertama dan cocok dengan bagian kedua. Jika demikian, ia mencatat sebagai pencocokan seluruh substring. Setelah proses ini selesai, string input dihasilkan oleh tata bahasa jika substring yang berisi seluruh string input dicocokkan dengan simbol awal.



Gambar skema algoritma CYK menerima sebuah string

BAB 2

Hasil Finite Automata dan Context-Free Grammar

Context-Free Grammar

S -> COMMENT	S -> TRY_CATCH_BODY
S -> FOR_LOOP	S -> CURFEW_OPEN
S -> WHILE_LOOP	S -> CURFEW_CLOSE
S -> LOOP_BODY	S -> CURFEW_CLOSE IF_METHOD
S -> LOOP_BODY CURFEW_CLOSE	S -> CURFEW_CLOSE ELSE_METHOD
S -> IF_METHOD	S -> CURFEW_CLOSE
S -> ELSE_METHOD	TRY_CATCH_METHOD
S -> COND_BODY	S -> CURFEW_CLOSE CATCH
S -> COND_BODY CURFEW_CLOSE	PAREN_OPEN OBJECT PAREN_CLOSE
S -> FUNCTION_METHOD	CURFEW_OPEN
S -> FUNCT_BODY	S -> CURFEW_CLOSE CATCH
S -> FUNCT_BODY CURFEW_CLOSE	PAREN_OPEN OBJECT PAREN_CLOSE
S -> SWITCH_METHOD	S -> CURFEW_CLOSE FINALLY
S -> CASE_METHOD	CURFEW_OPEN
S -> DEFAULT_METHOD	S -> CURFEW_CLOSE FINALLY
S -> CASE_BODY	S -> CURFEW_CLOSE FOR_LOOP
S -> TRY_CATCH_METHOD	S -> CURFEW_CLOSE WHILE_LOOP
S -> CATCH PAREN_OPEN OBJECT	S -> CURFEW_CLOSE
PAREN_CLOSE CURFEW_OPEN	FUNCTION_METHOD
S -> CATCH PAREN_OPEN OBJECT	S -> CURFEW_CLOSE SWITCH_METHOD
PAREN_CLOSE	S -> CURFEW_CLOSE VAR_OBJECT
S -> FINALLY CURFEW_OPEN	S -> CURFEW_CLOSE LET_OBJECT
S -> FINALLY	S -> CURFEW_CLOSE CONST_OBJECT
S -> THROW_OBJECT	S -> CURFEW_CLOSE OBJECT
S -> DELETE_OBJECT	SEMICOLON
	S -> CURFEW_CLOSE THROW_OBJECT
	S -> CURFEW_CLOSE DELETE_OBJECT

S -> MULTI_LINE_COMMENT_CLOSE
 S -> MULTI_LINE_COMMENT_CLOSE
 IF_METHOD
 S -> MULTI_LINE_COMMENT_CLOSE
 ELSE_METHOD
 S -> MULTI_LINE_COMMENT_CLOSE
 TRY_CATCH_METHOD
 S -> MULTI_LINE_COMMENT_CLOSE
 CATCH PAREN_OPEN OBJECT
 PAREN_CLOSE CURFEW_OPEN
 S -> MULTI_LINE_COMMENT_CLOSE
 CATCH PAREN_OPEN OBJECT
 PAREN_CLOSE
 S -> MULTI_LINE_COMMENT_CLOSE
 FINALLY CURFEW_OPEN
 S -> MULTI_LINE_COMMENT_CLOSE
 FINALLY
 S -> MULTI_LINE_COMMENT_CLOSE
 FOR_LOOP
 S -> MULTI_LINE_COMMENT_CLOSE
 WHILE_LOOP
 S -> MULTI_LINE_COMMENT_CLOSE
 FUNCTION_METHOD
 S -> MULTI_LINE_COMMENT_CLOSE
 SWITCH_METHOD
 S -> MULTI_LINE_COMMENT_CLOSE
 VAR_OBJECT
 S -> MULTI_LINE_COMMENT_CLOSE
 LET_OBJECT
 S -> MULTI_LINE_COMMENT_CLOSE
 CONST_OBJECT
 S -> MULTI_LINE_COMMENT_CLOSE
 OBJECT SEMICOLON
 S -> MULTI_LINE_COMMENT_CLOSE
 THROW_OBJECT
 S -> MULTI_LINE_COMMENT_CLOSE
 DELETE_OBJECT

S -> VAR_OBJECT
 S -> LET_OBJECT
 S -> CONST_OBJECT
 S -> OBJECT SEMICOLON
 COMMENT -> SINGLE_LINE_COMMENT
 OBJECT
 COMMENT ->
 MULTI_LINE_COMMENT_OPEN
 FUNCTION_METHOD -> FUNCTION NAME
 PAREN_OPEN PARAMETER
 PAREN_CLOSE CURFEW_OPEN
 FUNCTION_METHOD -> FUNCTION NAME
 PAREN_OPEN PARAMETER
 PAREN_CLOSE
 FUNCTION_METHOD -> FUNCTION NAME
 PAREN_OPEN PAREN_CLOSE
 CURFEW_OPEN
 FUNCTION_METHOD -> FUNCTION NAME
 PAREN_OPEN PAREN_CLOSE
 CURFEW_OPEN
 FUNCT_BODY -> OBJECT
 RETURN_METHOD
 FUNCT_BODY -> OBJECT SEMICOLON
 FUNCT_BODY -> RETURN_METHOD
 FUNCT_BODY -> FOR_LOOP
 FUNCT_BODY -> WHILE_LOOP
 FUNCT_BODY -> IF_METHOD
 FUNCT_BODY -> FUNCTION_METHOD
 FUNCT_BODY -> SWITCH_METHOD
 FUNCT_BODY -> TRY_CATCH_METHOD
 PARAMETER -> OBJECT
 PARAMETER -> PARAMETER COMMA
 PARAMETER
 PARAMETER -> DOT_DOT_DOT OBJECT

IF_METHOD -> IF PAREN_OPEN OBJECT
PAREN_CLOSE CURFEW_OPEN

IF_METHOD -> IF PAREN_OPEN OBJECT
PAREN_CLOSE

ELSE_METHOD -> ELSE IF_METHOD

ELSE_METHOD -> ELSE CURFEW_OPEN

ELSE_METHOD -> ELSE

SWITCH_METHOD -> SWITCH
PAREN_OPEN OBJECT PAREN_CLOSE
CURFEW_OPEN

SWITCH_METHOD -> SWITCH
PAREN_OPEN OBJECT PAREN_CLOSE

CASE_METHOD -> CASE INTEGER COLON

CASE_METHOD -> CASE STRING COLON

DEFAULT_METHOD -> DEFAULT COLON

CASE_BODY -> OBJECT EQUAL_SIGN
OBJECT SEMICOLON

CASE_BODY -> BREAK SEMICOLON

CASE_BODY -> IF_METHOD

CASE_BODY -> FOR_LOOP

CASE_BODY -> WHILE_LOOP

CASE_BODY -> SWITCH_METHOD

CASE_BODY -> TRY_CATCH_METHOD

COND_BODY -> OBJECT SEMICOLON

COND_BODY -> OBJECT BREAK
SEMICOLON

COND_BODY -> OBJECT CONTINUE
SEMICOLON

COND_BODY -> BREAK SEMICOLON

COND_BODY -> CONTINUE SEMICOLON

COND_BODY -> RETURN_METHOD

COND_BODY -> OBJECT SEMICOLON
BREAK SEMICOLON OBJECT SEMICOLON

COND_BODY -> OBJECT SEMICOLON
CONTINUE SEMICOLON OBJECT
SEMICOLON

COND_BODY -> BREAK SEMICOLON
OBJECT SEMICOLON

COND_BODY -> CONTINUE SEMICOLON
OBJECT SEMICOLON

COND_BODY -> IF_METHOD

COND_BODY -> FOR_LOOP

COND_BODY -> WHILE_LOOP

COND_BODY -> SWITCH_METHOD

COND_BODY -> TRY_CATCH_METHOD

FOR_LOOP -> FOR PAREN_OPEN OBJECT
SEMICOLON OBJECT SEMICOLON
OBJECT PAREN_CLOSE CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN
VAR_OBJECT OBJECT SEMICOLON
OBJECT PAREN_CLOSE CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN
LET_OBJECT OBJECT SEMICOLON
OBJECT PAREN_CLOSE CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN
CONST_OBJECT OBJECT SEMICOLON
OBJECT PAREN_CLOSE CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN OBJECT
IN OBJECT PAREN_CLOSE
CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN OBJECT
OF OBJECT PAREN_CLOSE
CURFEW_OPEN

FOR_LOOP -> FOR PAREN_OPEN
VAR_OBJECT OBJECT SEMICOLON
OBJECT PAREN_CLOSE

FOR_LOOP -> FOR PAREN_OPEN
LET_OBJECT OBJECT SEMICOLON
OBJECT PAREN_CLOSE

FOR_LOOP -> FOR PAREN_OPEN
 CONST_OBJECT OBJECT SEMICOLON
 OBJECT PAREN_CLOSE

 FOR_LOOP -> FOR PAREN_OPEN OBJECT
 SEMICOLON OBJECT SEMICOLON
 OBJECT PAREN_CLOSE

 FOR_LOOP -> FOR PAREN_OPEN OBJECT
 IN OBJECT PAREN_CLOSE

 FOR_LOOP -> FOR PAREN_OPEN OBJECT
 OF OBJECT PAREN_CLOSE

 WHILE_LOOP -> WHILE PAREN_OPEN
 OBJECT PAREN_CLOSE CURFEW_OPEN

 WHILE_LOOP -> WHILE PAREN_OPEN
 OBJECT PAREN_CLOSE

 LOOP_BODY -> OBJECT SEMICOLON

 LOOP_BODY -> BREAK SEMICOLON

 LOOP_BODY -> CONTINUE SEMICOLON

 LOOP_BODY -> FOR_LOOP

 LOOP_BODY -> WHILE_LOOP

 LOOP_BODY -> IF_METHOD

 LOOP_BODY -> SWITCH_METHOD

 LOOP_BODY -> TRY_CATCH_METHOD

 TRY_CATCH_METHOD -> TRY
 CURFEW_OPEN

 TRY_CATCH_METHOD -> TRY

 TRY_CATCH_BODY -> OBJECT
 SEMICOLON

 TRY_CATCH_BODY -> FOR_LOOP

 TRY_CATCH_BODY -> WHILE_LOOP

 TRY_CATCH_BODY -> IF_METHOD

 TRY_CATCH_BODY ->
 FUNCTION_METHOD

 TRY_CATCH_BODY -> SWITCH_METHOD

TRY_CATCH_BODY ->
 TRY_CATCH_METHOD

 THROW_OBJECT -> THROW OBJECT
 SEMICOLON

 DELETE_OBJECT -> DELETE OBJECT
 SEMICOLON

 SINGLE_LINE_COMMENT ->
 'SINGLE_LINE_COMMENT'

 MULTI_LINE_COMMENT_OPEN ->
 'MULTI_LINE_COMMENT_OPEN'

 MULTI_LINE_COMMENT_CLOSE ->
 'MULTI_LINE_COMMENT_CLOSE'

 IF -> 'IF'

 ELSE -> 'ELSE'

 PAREN_OPEN -> 'PAREN_OPEN'

 PAREN_CLOSE -> 'PAREN_CLOSE'

 OBJECT -> STRING

 OBJECT -> NUM

 OBJECT -> BOOL

 OBJECT -> NULL

 OBJECT -> OBJECT OBJECT

 OBJECT -> 'OBJECT'

 OBJECT -> PAREN_OPEN OBJECT
 PAREN_CLOSE

 OBJECT -> NAME

 OBJECT -> SIGN NAME

 OBJECT -> NOT OBJECT

 OBJECT -> ARRAY

 OBJECT -> OBJECT PAREN_OPEN
 PAREN_CLOSE DOT OBJECT
 PAREN_OPEN PAREN_CLOSE

 OBJECT -> OBJECT PAREN_OPEN OBJECT
 PAREN_CLOSE DOT OBJECT
 PAREN_OPEN PAREN_CLOSE

OBJECT -> OBJECT PAREN_OPEN OBJECT
PAREN_CLOSE DOT OBJECT
PAREN_OPEN OBJECT PAREN_CLOSE

OBJECT -> OBJECT PAREN_OPEN OBJECT
PAREN_CLOSE

OBJECT -> OBJECT PAREN_OPEN
PAREN_CLOSE

OBJECT -> OBJECT DOT OBJECT

OBJECT -> OBJECT DOT OBJECT
PAREN_OPEN PAREN_CLOSE

OBJECT -> OBJECT DOT OBJECT
PAREN_OPEN OBJECT PAREN_CLOSE

OBJECT -> OBJECT SEMICOLON

OBJECT -> OBJECT OP OBJECT

OBJECT -> NAME PAREN_OPEN
PARAMETER PAREN_CLOSE

OBJECT -> INCREMENT NAME

OBJECT -> DECREMENT NAME

OBJECT -> NAME INCREMENT

OBJECT -> NAME DECREMENT

OP -> COMPARISON_OP

OP -> ARITHMETIC_OP

OP -> BOOL_OP

OP -> ASSIGN_OP

OP -> BIT_OP

STRING -> DOUBLE_QUOTE OBJECT
DOUBLE_QUOTE

STRING -> SINGLE_QUOTE OBJECT
SINGLE_QUOTE

STRING -> SMART_QUOTE OBJECT
SMART_QUOTE

STRING -> DOUBLE_QUOTE
DOUBLE_QUOTE

STRING -> SINGLE_QUOTE
SINGLE_QUOTE

STRING -> SMART_QUOTE
SMART_QUOTE

NUM -> INTEGER

NUM -> SIGN INTEGER

NUM -> FLOAT

NUM -> SIGN FLOAT

COMPARISON_OP -> DOUBLE_EQUAL

COMPARISON_OP -> TRIPLE_EQUAL

COMPARISON_OP -> NOT_EQUAL

COMPARISON_OP ->
NOT_DOUBLE_EQUAL

COMPARISON_OP -> GREATER

COMPARISON_OP -> GREATER_EQUAL

COMPARISON_OP -> LESS

COMPARISON_OP -> LESS_EQUAL

COMPARISON_OP -> TERNARY

ARITHMETIC_OP -> PLUS

ARITHMETIC_OP -> MINUS

ARITHMETIC_OP -> DIVIDE

ARITHMETIC_OP -> MULTI

ARITHMETIC_OP -> EXPO

ARITHMETIC_OP -> MOD

ARITHMETIC_OP -> INCREMENT

ARITHMETIC_OP -> DECREMENT

BOOL_OP -> AND

BOOL_OP -> OR

BOOL_OP -> NULLISH

ASSIGN_OP -> EQUAL_SIGN

ASSIGN_OP -> ARITHMETIC_OP
 EQUAL_SIGN
 ASSIGN_OP -> BOOL_OP EQUAL_SIGN
 ASSIGN_OP -> BIT_OP EQUAL_SIGN
 ASSIGN_OP -> 'ASSIGN_OP'
 BIT_OP -> BITWISE_AND
 BIT_OP -> BITWISE_OR
 BIT_OP -> XOR
 BIT_OP -> LEFT_SHIFT
 BIT_OP -> RIGHT_SHIFT
 BIT_OP -> UNSIGNED_RIGHT_SHIFT
 INCREMENT -> PLUS PLUS
 DECREMENT -> MINUS MINUS
 RETURN_METHOD -> RETURN
 SEMICOLON
 RETURN_METHOD -> RETURN OBJECT
 SEMICOLON
 VAR_OBJECT -> VAR
 VAR_NAME_DECLARE SEMICOLON
 VAR_OBJECT -> VAR NAME_DECLARE
 SEMICOLON
 LET_OBJECT -> LET
 LET_NAME_DECLARE SEMICOLON
 LET_OBJECT -> LET NAME_DECLARE
 SEMICOLON
 CONST_OBJECT -> CONST
 CONST_NAME_DECLARE SEMICOLON
 NAME_DECLARE -> NAME
 NAME_DECLARE -> NAME EQUAL_SIGN
 OBJECT
 NAME_DECLARE -> NAME_DECLARE
 COMMA NAME_DECLARE

VAR_NAME_DECLARE ->
 VAR_NAME_DECLARE COMMA
 VAR_NAME_DECLARE
 VAR_NAME_DECLARE -> NAME
 EQUAL_SIGN OBJECT
 VAR_NAME_DECLARE -> NAME
 EQUAL_SIGN CURFEW_OPEN
 OBJECT_VAR CURFEW_CLOSE
 OBJECT_VAR -> NAME COLON OBJECT
 OBJECT_VAR -> OBJECT_VAR COMMA
 OBJECT_VAR
 LET_NAME_DECLARE ->
 LET_NAME_DECLARE COMMA
 LET_NAME_DECLARE
 LET_NAME_DECLARE -> NAME
 EQUAL_SIGN OBJECT
 LET_NAME_DECLARE -> NAME
 EQUAL_SIGN CURFEW_OPEN
 OBJECT_LET CURFEW_CLOSE
 OBJECT_LET -> NAME COLON OBJECT
 OBJECT_LET -> OBJECT_LET COMMA
 OBJECT_LET
 CONST_NAME_DECLARE ->
 CONST_NAME_DECLARE COMMA
 CONST_NAME_DECLARE
 CONST_NAME_DECLARE -> NAME
 EQUAL_SIGN OBJECT
 CONST_NAME_DECLARE -> NAME
 EQUAL_SIGN CURFEW_OPEN
 OBJECT_CONST CURFEW_CLOSE
 OBJECT_CONST -> NAME COLON OBJECT
 OBJECT_CONST -> OBJECT_CONST
 COMMA OBJECT_CONST
 FUNCTION -> 'FUNCTION'
 RETURN -> 'RETURN'
 BITWISE_AND -> 'BITWISE_AND'

BITWISE_OR -> 'BITWISE_OR'
LEFT_SHIFT -> 'LEFT_SHIFT'
RIGHT_SHIFT -> 'RIGHT_SHIFT'
UNSIGNED_RIGHT_SHIFT ->
'UNSIGNED_RIGHT_SHIFT'
XOR -> 'XOR'
MOD -> 'MOD'
AND -> 'AND'
OR -> 'OR'
EQUAL_SIGN -> 'EQUAL_SIGN'
NULLISH -> 'NULLISH'
DOUBLE_EQUAL -> 'DOUBLE_EQUAL'
TRIPLE_EQUAL -> 'TRIPLE_EQUAL'
NOT_EQUAL -> 'NOT_EQUAL'
NOT_DOUBLE_EQUAL ->
'NOT_DOUBLE_EQUAL'
GREATER -> 'GREATER'
GREATER_EQUAL -> 'GREATER_EQUAL'
LESS -> 'LESS'
LESS_EQUAL -> 'LESS_EQUAL'
TERNARY -> 'TERNARY'
PLUS -> 'PLUS'
MINUS -> 'MINUS'
DIVIDE -> 'DIVIDE'
MULTI -> 'MULTI'
INTEGER -> 'TYPE_INT'
FLOAT -> 'TYPE_FLOAT'
SIGN -> PLUS
SIGN -> MINUS
DOUBLE_QUOTE -> 'DOUBLE_QUOTE'
SINGLE_QUOTE -> 'SINGLE_QUOTE'

BOOL -> 'TRUE'
BOOL -> 'FALSE'
NULL -> 'NULL'
DOT -> 'DOT'
SQUARE_BRACKETS ->
'SQUARE_BRACKETS'
BREAK -> 'BREAK'
CONTINUE -> 'CONTINUE'
SEMICOLON -> 'SEMICOLON'
COLON -> 'COLON'
DOUBLE_EQUAL -> 'DOUBLE_EQUAL'
TYPE_INT -> 'TYPE_INT'
CURFEW_OPEN -> 'CURFEW_OPEN'
CURFEW_CLOSE -> 'CURFEW_CLOSE'
SWITCH -> 'SWITCH'
CASE -> 'CASE'
DEFAULT -> 'DEFAULT'
FOR -> 'FOR'
WHILE -> 'WHILE'
IN -> 'IN'
OF -> 'OF'
AS -> 'AS'
IS -> 'IS'
COMMA -> 'COMMA'
VAR -> 'VAR'
LET -> 'LET'
CONST -> 'CONST'
NAME -> 'NAME'
TRY -> 'TRY'
CATCH -> 'CATCH'

THROW -> 'THROW'

DELETE -> 'DELETE'

FINALLY -> 'FINALLY'

DOT_DOT_DOT -> 'DOT_DOT_DOT'

SMART_QUOTE -> 'SMART_QUOTE'

NOT -> 'NOT'

Finite Automata

```
# Finite Automata to check the variable name and operations
# JavaScript has only a few rules for variable names:
# 1. The first character must be a letter or an underscore (_).
# 2. You can't use a number as the first character.
# 3. The rest of the variable name can include any letter, any number, or the underscore.
# You can't use any other characters, including spaces, symbols, and punctuation marks.
# 4. As with the rest of JavaScript, variable names are case sensitive.
# 5. There's no limit to the length of the variable name.

import re

def FA(readstring):
    # Inisialization
    countvarerror = 0
    hasil = []
    hasil1 = []
    hasil2 = []

    # Defining the DFA States
    hurufkecil = [chr(c) for c in range(97,123)]
    hurufkapital = [chr(c) for c in range(65,91)]
    angka = [chr(c) for c in range(48,58)]
    comment = ['/', '/', '*', '/', '*']
    batas = ['&', '?', '@', '^', '~']
    pembanding = ['=', '>', '<', '!']
    aritmetik = ['+', '-', '/', '*', '%']
    pembuka = ['{', '[', '(', '']
    penutup = ['}', ']', ')', '']
    dfa = {0:{}, 1:{}, 2:{}, 3:{}, 4:{}, 5:{}}

    # State 0
    for x in hurufkecil:
        dfa[0][x] = 2
    for x in hurufkapital:
        dfa[0][x] = 2
    for x in angka:
        dfa[0][x] = 3
    for x in comment:
        dfa[0][x] = 1
    for x in batas:
        dfa[0][x] = 4
    for x in pembanding:
        dfa[0][x] = 0
    for x in pembuka:
        dfa[0][x] = 0
    for x in penutup:
        dfa[0][x] = 0
    for x in aritmetik:
        dfa[0][x] = 0
    dfa[0]['_'] = 2
```

```

dfa[0]['$'] = 2
dfa[0]['.'] = 0
dfa[0][':'] = 0

# State 1
for x in hurufkecil:
    dfa[1][x] = 1
for x in hurufkapital:
    dfa[1][x] = 1
for x in angka:
    dfa[1][x] = 1
for x in comment:
    dfa[1][x] = 0
for x in batas:
    dfa[1][x] = 1
for x in pembanding:
    dfa[1][x] = 1
for x in pembuka:
    dfa[1][x] = 1
for x in penutup:
    dfa[1][x] = 1
for x in aritmetik:
    dfa[1][x] = 1
dfa[1]['_'] = 1
dfa[1]['$'] = 1
dfa[1]['.'] = 1
dfa[1][':'] = 1

# State 2
for x in hurufkecil :
    dfa[2][x] = 2
for x in hurufkapital:
    dfa[2][x] = 2
for x in angka:
    dfa[2][x] = 2
for x in comment:
    dfa[2][x] = 1
for x in batas:
    dfa[2][x] = 4
for x in pembanding:
    dfa[2][x] = 0
for x in pembuka:
    dfa[2][x] = 0
for x in penutup:
    dfa[2][x] = 0
for x in aritmetik:
    dfa[2][x] = 0
dfa[2]['_'] = 2
dfa[2]['$'] = 2
dfa[2]['.'] = 0
dfa[2][':'] = 0

# State 3
for x in hurufkecil :
    dfa[3][x] = 4
for x in hurufkapital:
    dfa[3][x] = 4
for x in angka:
    dfa[3][x] = 3
for x in comment:
    dfa[3][x] = 0

```

```

for x in batas:
    dfa[3][x] = 4
for x in pembanding:
    dfa[3][x] = 0
for x in pembuka:
    dfa[3][x] = 4
for x in penutup:
    dfa[3][x] = 0
for x in aritmetik:
    dfa[3][x] = 0
dfa[3]['_'] = 4
dfa[3]['$'] = 4
dfa[3]['.'] = 5
dfa[3][':'] = 0

# State 5
for x in hurufkecil :
    dfa[5][x] = 4
for x in hurufkapital:
    dfa[5][x] = 4
for x in angka:
    dfa[5][x] = 5
for x in comment:
    dfa[5][x] = 0
for x in batas:
    dfa[5][x] = 4
for x in pembanding:
    dfa[5][x] = 0
for x in pembuka:
    dfa[5][x] = 4
for x in penutup:
    dfa[5][x] = 4
for x in aritmetik:
    dfa[5][x] = 0
dfa[5]['_'] = 4
dfa[5]['$'] = 4
dfa[5]['.'] = 5
dfa[5][':'] = 0

# Defining how a combination accepted
def accepts(transition, start, s):
    states = start
    for char in s:
        try:
            states = transition[states][char]
        except KeyError:
            return False
    return (states != 4)

for i in readstring:
    thisstring = re.sub('&&','=',i)
    hasil.append(thisstring)
for j in hasil:
    thisstring1 = re.sub('\|\\|','=',j)
    hasil1.append(thisstring1)
for k in hasil1:
    thisstring2 = re.sub('\s+',' ',k)
    hasil2.append(thisstring2)
for l in range(len(hasil2)):
    if (accepts(dfa,0,hasil2[l])):
        consider = 'Accepted'

```



```
    else:
        consider = 'Rejected'
        countvarerror += 1
        print(hasil[1], consider)

    return consider, hasil, hasil1, hasil2
```

BAB 3

Implementasi dan Pengujian

A. Implementasi pada *Source Code*

Program yang kami buat terdiri atas tujuh file utama dan beberapa file input untuk pada folder test yang kami gunakan sebagai sampel uji pengetesan. Secara spesifik, program yang kami buat terdiri atas lima file python dengan fungsionalnya masing-masing dan dua file txt yang merupakan CFG dan CNF yang menggerakkan program kami. File utama yang memuat keseluruhan alur program ada pada main.py. File ini memerlukan import dari file LexerGrammar sebagai tempat dilakukannya *lexing* untuk mengubah input yang terbaca menjadi sebuah susunan objek yang dapat di-*parse*, file CYK yang menerapkan dan mengimplementasikan algoritma Cocke-Younger-Kasami dalam sebuah *source code*, file FA yang digunakan sebagai validator sintaks masing-masing operator termasuk validator untuk nama variabel dan objek seperti nama fungsi, dan module time yang memungkinkan kami untuk melakukan perhitungan *run time* eksekusi dari program *parsing* yang kami buat. Kami juga mengimport module os.path untuk dengan mudah melakukan validasi apakah file yang diinputkan user ada di dalam *directory* atau tidak.

Penjelasan masing-masing program secara detail dapat dinyatakan sebagai berikut :

1. Converter.py

No.	Fungsi/Prosedur	Tujuan
1.	read_grammar(grammar_file)	Membaca grammar dari sebuah file txt. Dalam hal ini, file CFG.txt
2.	add_rule(rule)	Menambahkan aturan (<i>rule</i>) <i>grammar</i> ke dalam sebuah <i>rule dictionary</i> yang merupakan variabel global
3.	convert_grammar(grammar)	Merupakan fungsi utama dalam file ini. Berfungsi mengubah grammar yang terdapat pada CFG.txt menjadi sebuah <i>Chomsky Normal Form (CNF)</i> pada file CNF.txt

2. CYK.py

No.	Fungsi/Prosedur	Tujuan
Class Node		
1.	<code>__init__(self, symbol, child1, child2=None)</code>	Konstruktor yang membuat sebuah simbol dengan dua buah <i>child</i>
2.	<code>__repr__(self)</code>	Mengubah representasi <i>object</i> menjadi <i>string</i> untuk memudahkan proses CYK
Class Parser		
1.	<code>__init__(self, grammar)</code>	Konstruktor tabel <i>parsing</i> , <i>production rules</i> , <i>grammar</i> , <i>sentences</i> , dan string input
2.	<code>__call__(self, sentence, parse=False)</code>	Memanggil hasil konstruksi untuk melanjutkan pemrosesan menggunakan CYK
3.	<code>grammarFile(self, grammar)</code>	Membaca file manapun dari string
4.	<code>parsing(self)</code>	Melakukan <i>parsing</i> hasil input file
5.	<code>print_tree(self, output = True)</code>	Menghasilkan output bagian yang salah pada hasil algoritma CYK apabila terjadi kesalahan sintaks
Without Class		
1.	<code>generate_tree(node)</code>	Mengenerasi binay tree hasil parse menggunakan algoritma CYK yang mungkin berdasarkan rules yang tersedia

3. FA.py

No.	Fungsi/Prosedur	Tujuan
1.	<code>FA(readstring)</code>	Membaca dan melakukan pemrosesan pada setiap variable dan sintaks dimana FA ini digunakan sehingga dapat diperiksa apakah sintaks sudah sesuai dengan aturan yang benar.

4. LexerGrammar.py

No.	Fungsi/Prosedur	Tujuan
Class Token(object)		
1.	<code>__init__(self, type, val, pos)</code>	Konstruktor proses tokenisasi berdasarkan tipe data, nilai, dan posisi
2.	<code>__str__(self)</code>	Melakukan perubahan hasil bacaan konstruksi menjadi string
3.	<code>__repr__(self)</code>	Membangkitkan representasi tipe data token
Class LexerError(Exception)		
1.	<code>__init__(self, pos)</code>	Melakukan handle pada hasil lexer yang tidak terdeteksi
Class Lexer(object)		
1.	<code>__init__(self, rules, skip_whitespace=True)</code>	Inisialisasi pembacaan kumpulan aturan menjadi beberapa token
2.	<code>input(self, buf)</code>	Melakukan pembacaan terhadap sebuah input string yang didapat dari file per baris menggunakan fungsi <code>readlines()</code>
3.	<code>token(self)</code>	Proses tokenisasi dengan rules yang telah dibuat dan disajikan
Without Class		
1.	<code>rules</code>	Berisi kumpulan <i>rules</i> yang digunakan untuk membantu proses perubahan isi file yang terbaca menjadi token

5. main.py

Program utama yang memanggil semua file yang telah didefinisikan sebelumnya. Skema pemrosesan yang dilakukan dimulai dengan melakukan pembacaan grammar pada CFG.txt, kemudian diubah menjadi CNF.txt agar dapat diproses menggunakan algoritma CYK. Proses pembacaan file dibaca per line sehingga memudahkan untuk melakukan identifikasi lokasi error dari file yang diinputkan ke dalam main program. Selanjutnya akan dicek berdasarkan aturan lexer yang beragam, mulai dari conditional (IF, ELSE,

SWITCH, CASE), fungsi (FUNCTION), comment (SINGLE_LINE_COMMENT, MULTI_LINE_COMMENT), loop (FOR, WHILE, BREAK, CONTINUE) dan lain sebagainya sesuai dengan aturan dari masing-masing fungsi. Selain itu kamu juga menggunakan konsep level untuk mengatur pemosisian conditional dan function dalam sebuah program dan representasi stack of curfew untuk mengidentifikasi posisi dan indentasi dari setiap program yang dibuat. Setelah semua pemrosesan tersebut dilakukan, program akan mengeluarkan hasil verdict dari file program yang diinputkan. Jika program dalam file tersebut valid secara sintaks, maka akan dikeluarkan “Accepted” dengan warna hijau, sebaliknya jika tidak maka akan dikeluarkan “Syntax Error” lengkap dengan letak dan detail kesalahan pada sintaks.

B. Pengujian

1. Tampilan Program Utama

```

$$$$\      === TUGAS BESAR TBFO IF2124 ===      $$\      $$\
\_ $$ |      \_ |      $$ |
  $$ |$$$$$\ $$\  $$$\$$$$\  $$$\$$$$\  $$$\$$$$\  $$$\ $$$\$$$$\$$$$\
  $$ | \_ $$$\ $$\  $$$\ $$$\  $$$\ $$$\  $$$\ $$$\  $$$\ $$$\  _|
$$\  $$ |$$$$$\ $$$\ $$$\ /$$$$$\ $$$\ $$$\ /  $$$\ $$$\ /  $$$\ $$$\
$$ |  $$ $$$\ $$$\ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$
\$$$$\ \$$$$\ \$$$ / \$$$$\ $$$\ $$$\ \$$$$\ $$$\ \$$$ |  $$$\ $$$\ $$$\
\_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\
      P A R S E R      -- Made by LSN      $$\
      \_ |      $$$\
      \_ |

Insert file name (.js): 

```

Tampilan awal program utama

2. Pemrosesan Program Input

```

$$$$\      === TUGAS BESAR TBFO IF2124 ===      $$\      $$\
\_ $$ |      \_ |      $$ |
  $$ |$$$$$\ $$$\  $$$\$$$$\  $$$\$$$$\  $$$\$$$$\  $$$\ $$$\$$$$\$$$$\
  $$ | \_ $$$\ $$\  $$$\ $$$\  $$$\ $$$\  $$$\ $$$\  $$$\ $$$\  _|
$$\  $$ |$$$$$\ $$$\ $$$\ /$$$$$\ $$$\ $$$\ /  $$$\ $$$\ /  $$$\ $$$\
$$ |  $$ $$$\ $$$\ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$ /$$$
\$$$$\ \$$$$\ \$$$ / \$$$$\ $$$\ $$$\ \$$$$\ $$$\ \$$$ |  $$$\ $$$\ $$$\
\_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\ \_ $$$\
      P A R S E R      -- Made by LSN      $$\
      \_ |      $$$\
      \_ |

Insert file name (.js): test/dummy.js
There's no such file in directory!

```

Tampilan jika program tidak ada pada directory

```

$$$$\          === TUGAS BESAR TBFO IF2124 ===          $$\          $$\
\_ $$ |          \_ |          $$ |
  $$ |$$$$$\$ $\$ \$$$$$\$ \$$$$$\$ \$$$$$\$ \$$$$$\$ \$$$$$\$ \$$$$$\$ \
  $$ |\_ $$$$\$ \  $$ \_ $$$$\$ \_ $$$ \_ $$$ \_ $$$ \_ $$$ \_ $$$ \_ |
  $$\  $$ |$$$$$$$ \$$\$$$ /$$$$$$$ \$$$$$$$ \$$$ /  $$$ \_ $$$ $$$ /  $$$ |$$$ |
  $$ |  $$ $$$ \_ $$$ \_ $$$ /$$$ \_ $$$ |\_ $$$\$$$ |  $$$ |  $$$ $$$ | $$$ |$$$ \
 \$$$$$$$ \$$$$$$$ | \$$$ /$$$ \_ $$$ |\_ $$$\$$$ |  $$$ |  $$$ $$$ | \$$$$$ |
 \_ $$$\ / \_ $$$\ | \_ $$$\ \_ $$$\ / \_ $$$\ \_ $$$\ |  $$$ \_ $$$\ / \_ $$$\
                                     $$$ |
                                     $$$ |
                                     \_ |
                                     $$$ |
                                     $$$ |
                                     \_ |

      P      A      R      S      E      R      -- Made by LSN

```

Insert file name (.js): test/coba.js

```

===== PARSING =====

console.log("Hello World");

===== VERDICT =====

Compiling 1 line(s) of code from test/coba.js....
Get a result....

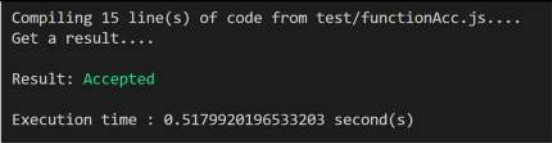
Result: Accepted

Execution time : 0.21103310585021973 second(s)

```

Tampilan jika program ada pada directory, program akan melakukan parsing, menampilkan isi file, mencatat jumlah *line*, memberikan *result*, dan mengeluarkan *execution time*

3. Test Case Function

Program	Hasil
Nama File : funcAcc.js	
<pre> function Product(a,b,c,d) { let x = a+b; x -= c; if (x == 1) { return a; } return x; } function toCelsius (fahrenheit) { return fahrenheit; } document.getElementById("demo").innerHTML ML = toCelsius; Product(a,b,c,e); toCelsius(20); </pre>	
Analisis :	
<p>File diterima karena sesuai dengan sintaks fungsi yang seharusnya. Posisi return berada di dalam function. Output sesuai dengan yang diharapkan.</p>	

Nama File : funcReject1.js	
<pre> 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"; } </pre>	<pre> Compiling 13 line(s) of code from test/functionReject1.js.... Get a result.... Result: Syntax Error at line 2: >> return 0; Readed: RETURN TYPE_INT SEMICOLON Execution time : 0.07651615142822266 second(s) </pre>
Analisis : File tidak diterima karena tidak sesuai dengan sintaks. Tidak dapat meletakkan return jika tidak berada dalam fungsi atau loop. Output sesuai dengan yang diharapkan, terdapat kesalahan pada line ke-2.	
Nama File : funcReject2.js	
<pre> let x = 7; function Product(a,b,c,d) { let x = a+b; x -= c; if (x == 1) { return a; } while (true) { if (x == 2) { break; } else { x++; } } return x; } return x; </pre>	<pre> Compiling 17 line(s) of code from test/functionReject2.js.... Get a result.... Result: Syntax Error at line 17: >> return x; Readed: RETURN NAME SEMICOLON Execution time : 0.6150343418121338 second(s) </pre>
Analisis : File tidak diterima karena tidak sesuai dengan sintaks. Posisi return berada di luar fungsi yang telah didefinisikan sebelumnya. Output sesuai dengan yang diharapkan, terdapat kesalahan pada line ke-17.	

4. Test Case Conditional if-else

Program	Hasil
Nama File : ifAcc.js	

<pre> if (x == 0) { print("No1"); } else if (x + 4 == 1) { if (true) { print(3); } else { print(2); } } else if (x == 32) { for (let i=0; i<10; i++) { print(i); } } else { print("Momen"); } </pre>	<pre> Compiling 17 line(s) of code from ./test/ifAcc.js.... Get a result.... Result: Accepted Execution time : 1.949000358581543 second(s) </pre>
---	---

Analisis :

File diterima karena sesuai dengan sintaks, posisi print berada didalam percabangan dan menghasilkan output yang sesuai dengan yang diharapkan.

Nama File : ifReject.js

<pre> function sudahlelahmonangis(x) { // iseng nambahin single line comment if (x > 0) { if (x > 3) { return "Haha"; } } else { return 0; } else { return "Lelah"; } } </pre>	<pre> Compiling 14 line(s) of code from ./test/ifReject.js.... Get a result.... Result: Syntax Error at line 11: >> else { Readed: ELSE CURFEW_OPEN Execution time : 0.1829972267150879 second(s) </pre>
---	---

Analisis :

File tidak diterima karena tidak sesuai dengan sintaks, else kedua didefinisikan diluar if, sehingga menghasilkan output error pada line 11.

5. Test Case pada Spesifikasi Tugas Besar

Program	Hasil
Nama File : inputAcc.js	
<pre> function do_something(x) { // This is a sample comment if (x == 0) { return 0; } else if (x + 4 == 1) { if (true) { </pre>	<pre> Compiling 16 line(s) of code from ./test/inputAcc.js.... Get a result.... Result: Accepted Execution time : 0.35799646377563477 second(s) </pre>

<pre> return 3; } else { return 2; } } else if (x == 32) { return 4; } else { return "Momen"; } </pre>	
--	--

Analisis :

File diterima karena sesuai dengan sintaks, setiap kondisi pada percabangan diberikan tanda kurung untuk menghindari kebingungan pada program. Output memberikan hasil sesuai dengan yang diharapkan.

Nama File : inputReject.js

<pre> function do_something(x) { // This is a sample multiline 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"; } } </pre>	<pre> Compiling 16 line(s) of code from ./test/inputReject.js.... Get a result.... Result: Syntax Error at line 5: >> } else if x + 4 == 1 { Readed: CURFEM_CLOSE ELSE IF NAME PLUS TYPE_INT DOUBLE_EQUAL TYPE_INT CURFEM_OPEN Execution time : 0.2059931755065918 second(s) </pre>
--	--

Analisis :

File tidak diterima karena tidak sesuai dengan sintaks, perhatikan pada line 5 bahwa kondisi percabangannya tidak menggunakan tanda kurung yang menyebabkan terjadinya kebingungan pada program. Outputnya menghasilkan error pada line tersebut.

6. Test Case Loop

Program	Hasil
Nama File : loopAcc.js	
<pre> let str = ''; for (let i = 0; i < 9; i++) { str = str + i; while (str != 1){ i++; } } function a() { </pre>	<pre> Compiling 20 line(s) of code from ./test/loopAcc.js.... Get a result.... Result: Accepted Execution time : 2.092029333114624 second(s) </pre>

<pre> test++; if (test < 10) { console.log(str); } else { test++; } } break; } while (i<10) { i++; } </pre>	
<p>Analisis :</p> <p>File diterima karena sesuai dengan sintaks, perhatikan bahwa integer i diberikan definisi dan batasan yang jelas sehingga dapat dieksekusi dengan baik dan memberikan output yang sesuai dengan harapan.</p>	
<p>Nama File : loopReject.js</p>	
<pre> for (let i) { let x; x += i; break; } </pre>	<pre> Compiling 5 line(s) of code from ./test/loopReject.js.... Get a result.... Result: Syntax Error at line 1: >> for (let i) { Readed: FOR PAREN_OPEN LET NAME PAREN_CLOSE CURFEW_OPEN Execution time : 0.039999961853027344 second(s) </pre>
<p>Analisis :</p> <p>File tidak diterima karena tidak sesuai dengan sintaks, pada variabel i tidak diberikan definisi dan batasan yang jelas sehingga program tidak dapat dieksekusi dengan baik yang mana menghasilkan ouput error pada line 1.</p>	

7. Test Case Conditional switch-case

Program	Hasil
<p>Nama File : switchAcc.js</p>	
<pre> switch(new Date().getDay()) { case 0: day = "monday"; break; case 1: day = "Monday"; break; case 2: day = "Tuesday"; break; case 3: day = "Wednesday"; break; case 4: day = "Thursday"; </pre>	<pre> Compiling 24 line(s) of code from ./test/switchAcc.js.... Get a result.... Result: Accepted Execution time : 0.3290293216705322 second(s) </pre>

<pre> break; case 5: day = "Friday"; break; case 6: day = "Saturday"; default: day = "Whatever"; } </pre>	
<p>Analisis :</p> <p>File diterima karena ditulis sesuai sintaks yang mana fungsi switch dan setiap casenya ditulis dengan baik sehingga memberikan output yang sesuai dengan yang diharapkan.</p>	
<p>Nama File : switchReject.js</p>	
<pre> switch (new Date().getDay()) case 0: // baca buku break; case 1: day = "Monday"; break; case 2: day = "Tuesday"; break; case 3: day = "Wednesday"; break; case 4: day = "Thursday"; break; case 5: day = "Friday"; break; case 6: day = "Saturday"; } </pre>	<div data-bbox="873 680 1416 903"> <pre> ===== VERDICT ===== Compiling 22 line(s) of code from ./test/switchReject.js.... Get a result.... Result: Syntax Error at line 2: >> case 0: Readed: SWITCH PAREN_OPEN NAME NAME PAREN_OPEN PAREN_CLOSE DO Execution time : 0.1719954013824463 second(s) </pre> </div> <p>* Terpotong supaya masih bisa terbaca di laporan</p>
<p>Analisis :</p> <p>File tidak diterima karena tidak sesuai dengan sintaks, perhatikan bahwa fungsinya tidak dibuka dengan tanda kurawal sebagaimana mestinya, sehingga menghasilkan ouput error pada line 2.</p>	

8. Test Case throw-delete

Program	Hasil
<p>Nama File : throwDeleteAcc.js</p>	
<pre> if (err) { throw err; } else { delete console.log(diaw); } </pre>	<div data-bbox="873 1738 1416 1869"> <pre> Compiling 5 line(s) of code from ./test/throwDeleteAcc.js.... Get a result.... Result: Accepted Execution time : 0.17899155616760254 second(s) </pre> </div>

Analisis : File dapat diterima karena sudah sesuai dengan sintaks, terlihat bahwa setiap kondisi percabangan memberikan hasil throw dan juga delete yang sesuai sehingga memberikan output hasil yang diharapkan.	
Nama File : throwDeleteReject.js	
<pre>if (err) { throw err; } else { delete; }</pre>	
Analisis : File tidak diterima karena tidak sesuai dengan sintaks, terlihat pada line 4 memberikan output error karena pada kondisi percabangan tersebut memberikan hasil delete yang tidak sesuai.	

9. Test Case try-catch

Program	Hasil
Nama File : tryCatchTestAcc.js	
<pre>try { adddler("Test haslo hdsajdw"); if (i > 0) { print("test"); } } catch (e) { document.getElementById("demo").inn erHTML = err.message; } finally { let i = 0; while (i < 10) { console.log("Hello"); } for (i=0; i<10; i++) { console.log("Hello"); } }</pre>	
Analisis : File dapat diterima karena sudah sesuai dengan sintaks yaitu baik fungsi try maupun catch diberikan kondisi yang jelas sehingga dapat dieksekusi dengan baik, serta memberikan hasil yang sesuai denan harapan.	
Nama File : tryCatchTestReject.js	

<pre> try { addAlert("Test haslo hdsajdw"); if (i > 0) { print("test"); } } catch () { document.getElementById("demo").innerHTMLHTML = err.message; } finally { let i = 0; while (i < 10) { console.log("Hello"); } for (i=0; i<10; i++) { console.log("Hello"); } } </pre>	<pre> Compiling 17 line(s) of code from ./test/tryCatchTestReject.js.... Get a result.... Result: Syntax Error at line 6: >> } catch () { Readed: CURFEW_CLOSE CATCH PAREN_OPEN PAREN_CLOSE CURFEW_OPEN Execution time : 0.3079829216003418 second(s) </pre>
--	---

Analisis :

File tidak diterima karena tidak sesuai dengan sintaks, yaitu pada line 16 tepatnya pada fungsi catch tidak diberikan kondisi yang jelas sehingga menghasilkan output error.

10. Test Case var-let-const

Program	Hasil
Nama File : varLetConstAcc.js	
<pre> let x,y=0,c; const e = 'a', b='b'; var i; var a = 10; function f(){ console.log(a); } f(); console.log(a); function f() { // It can be accessible any // where within this function var a = 10; console.log(a); } f(); // A cannot be accessible // outside of function console.log(a); let a = 10; function f() { if (true) { let b = 9; if (b==9) { // It prints 9 </pre>	<pre> Compiling 48 line(s) of code from ./test/varLetConstAcc.js.... Get a result.... Result: Accepted Execution time : 0.4889960289001465 second(s) </pre>

<pre> console.log(b); } } // It gives error as it // defined in if block console.log(b); } f(); // It prints 10 console.log(a); let a = 10; // It is not allowed let a = 10; // It is allowed a = 10; </pre>	
--	--

Analisis :

File diterima karena sudah sesuai dengan sintaks yaitu setiap fungsi var, let dan const diberikan definisi dan batasan yang jelas sehingga dapat dieksekusi dengan baik dan memberikan output yang sesuai dengan harapan.

Nama File : varLetConstReject.js

<pre> let x = 1; var i = 2,y; const a=1,b,c; </pre>	<pre> Compiling 4 line(s) of code from ./test/varLetConstReject.js.... Get a result.... Result: Syntax Error at line 4: >> const a=1,b,c; Readed: CONST NAME EQUAL_SIGN TYPE_INT COMMA NAME COMMA NAME SEMICOLON Execution time : 0.21799659729083906 second(s) </pre>
--	---

Analisis :

File tidak dapat diterima karena tidak sesuai dengan sintaks. Pendefinisian const hanya berupa variabel dan tidak di-assign dengan sebuah nilai sehingga menghasilkan error pada line ke 4

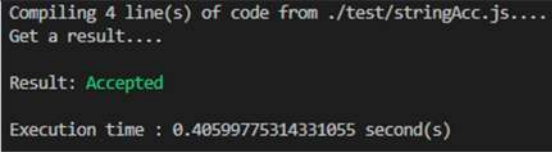

11. Test Case Variable

Program	Hasil
Nama File : variablesReject.js	
<pre> let 2pk = 1; </pre>	<pre> Compiling 1 line(s) of code from ./test/variablesReject.js.... Get a result.... Result: Syntax Error at line 1: >> let 2pk = 1; Readed: LET TYPE_INT NAME EQUAL_SIGN TYPE_INT SEMICOLON Execution time : 0.07199931144714355 second(s) </pre>

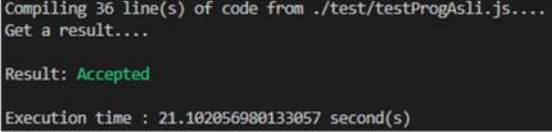
Analisis :

File tidak dapat diterima karena tidak sesuai dengan sintaks karena dalam JavaScript nama variabel tidak diperbolehkan didahului angka. Program memberikan output error pada line 1.

12. Test Case String

Program	Hasil
Nama File : stringAcc.js	
<pre>console.log('Test'Halo'); console.log("Test\"Halo"); console.log('Test\\'Halo'); console.log("Test'Halo");</pre>	 <pre>Compiling 4 line(s) of code from ./test/stringAcc.js.... Get a result.... Result: Accepted Execution time : 0.40599775314331055 second(s)</pre>
Analisis :	
File dapat diterima karena sudah sesuai dengan sintaks, yaitu penulisan string yang diawali dan diakhiri dengan tanda kutip yang baik sehingga memberikan output yang sesuai dengan harapan.	
Nama File : stringReject.js	
<pre>// console.log('Test'Halo'); console.log("Test\"Halo"); console.log('Test\\'Halo);</pre>	 <pre>Compiling 3 line(s) of code from ./test/stringReject.js.... Get a result.... Result: Syntax Error at line 2: >> console.log("Test\"Halo"); Readed: NAME DOT NAME PAREN_OPEN DOUBLE_QUOTE NAME DOUBLE_QUOTE NAME DOUBLE_QUOTE NAME NAME NAME</pre>
Analisis :	
File tidak dapat diterima karena tidak sesuai dengan sintaks, yaitu pada line 2 tanda kutip pada string tidak ditulis dengan benar. Selain itu tanda kutip bukan merupakan sebuah <i>escape character</i> sehingga memberikan hasil error.	

13. Test Case Program Lengkap

Program	Hasil
Nama File : testProgAsli.js	
<pre>// program to solve quadratic equation let root1, root2; // take input from the user let a = prompt("Enter the first number : "); let b = prompt("Enter the second number : "); let c = prompt("Enter the third number : "); // calculate discriminant let discriminant = b * b - 4 * a * c;</pre>	 <pre>Compiling 36 line(s) of code from ./test/testProgAsli.js.... Get a result.... Result: Accepted Execution time : 21.102056980133057 second(s)</pre>

```

// condition for real and different
roots
if (discriminant > 0) {
    root1 = (-b +
Math.sqrt(discriminant)) / (2 * a);
    root2 = (-b -
Math.sqrt(discriminant)) / (2 * a);

    // result
    console.log('The roots of quadratic
equation are root1 dan root2');
}

// condition for real and equal roots
else if (discriminant == 0) {
    root1 = root2 = -b / (2 * a);

    // result
    console.log(`The roots of quadratic
equation are root1 and root2`);
}

// if roots are not real
else {
    let realPart = (-b / (2 *
a)).toFixed(2);
    let imagPart = (Math.sqrt(-
discriminant) / (2 * a)).toFixed(2);

    // result
    console.log('The roots of quadratic
equation are i');
}

```

Analisis :

File dapat diterima karena sudah ditulis sesuai sintaks dan secara keseluruhan grammarnya telah didefinisikan dan dihandle dengan baik sehingga memberikan output yang sesuai dengan yang diharapkan.

BAB 4

Penutup

A. Kesimpulan

Kesimpulan yang bisa ditarik dari tugas besar Teori Bahasa Formal dan Otomata ini adalah sebagai berikut :

- a. Kelompok kami telah berhasil membuat FA dan CFG untuk mengecek string apakah diterima oleh sintaks JavaScript.
- b. Kami berhasil membuat program dengan JavaScript yang memanfaatkan FA dan CFG yang kami buat dengan menggunakan bantuan beberapa fungsi yang telah dipaparkan diatas.
- c. Sintaks dan aturan program dari bahasa pemrograman JavaScript yang relative sulit membuat algoritma CYK yang dibuat kurang dapat mencakup seluruh kemungkinan kasus pemosisian, indentasi, dan aturan agar sintaks dapat dianggap valid.
- d. Terlepas dari CYK yang terbuat, Program ini dapat berjalan dengan cukup baik dengan adanya LexerGrammar yang melakukan parsing bahasa pemrograman sesuai dengan rules yang dibuat.
- e. Hasil eksekusi sangat tergantung pada CFG yang dibuat dan CNF hasil konversi yang perlu cukup detail untuk menampung semua kemungkinan yang mungkin.
- f. Tingkat kompleksitas program akan mempengaruhi lama waktu eksekusi parsingnya.

Secara umum program yang kami buat berjalan dengan baik. Akan tetapi, masih terdapat beberapa kekurangan dalam eksekusi sehingga masih perlu adanya perbaikan dan optimasi untuk menghasilkan program yang lebih baik.

B. Saran

Algoritma CYK ini merupakan sebuah algoritma yang cukup tepat jika digunakan untuk melakukan parsing sebuah program untuk mengecek apakah sudah sesuai dengan aturan sintaks yang telah ada. Akan tetapi, dengan keterbatasan fungsi, pembacaan, dan waktu eksekusi yang relatif lama untuk program yang memiliki kompleksitas tinggi, perlu adanya pengembangan metode yang lebih efektif untuk menangani hal ini. Terlepas dari

permasalahan algoritma, kami juga belum dapat mendalami masing-masing komponen yang digunakan untuk menyusun sebuah program *parsing* ini sehingga untuk beberapa hal, hasilnya masih kurang maksimal. Selain itu, permasalahan yang kami buat adalah kami tidak melakukan validasi tipe data sejak awal, sehingga pemrosesan untuk *debugging* menjadi lebih sulit. Oleh sebab itu, sangat disarankan untuk melakukan pengembangan lebih lanjut mengenai program yang telah dibuat ini sehingga bisa menghasilkan *parser* yang lebih sempurna.

C. Lampiran

1. Link Repository Github

<https://github.com/mikeleo03/TubesTBFO>

2. Pembagian Tugas

Anggota	NIM	Tugas
Michael Leon Putra Widhi	13521108	Membuat CFG, Membuat dan melakukan konversi CFG ke CNF, Membuat algoritma CYK, Membuat implementasi <i>lexer</i> , Membuat FA, Membuat dasar program utama, Membuat <i>test case</i> , Membantu menyusun laporan
Satria Octavianus Nababan	13521168	Membuat CFG, Menyusun laporan, Melakukan <i>testing</i> , Membuat <i>test case</i>
Nathan Tenka	13521172	Membuat CFG, Membuat <i>test case</i> , Melengkapi penanganan <i>edge case</i> pada <i>lexer</i> , Melakukan penanganan kasus umum dan <i>edge case</i> di program utama