LAPORAN TUGAS BESAR

IF2124 TEORI BAHASA FORMAL DAN OTOMATA

Disusun oleh:

13521045 – Fakhri Muhammad Mahendra

13521089 – Kenneth Ezekiel Suprantoni

13521101 – Arsa Izdihar Islam

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2022

Daftar Isi

Bab I
Teori Dasar

A. Teori Dasar FA

Finite Automata, atau biasa disebut juga *Finite State Automata* adalah sebuah model matematika untuk komputasi. FA adalah sebuah mesin abstrak yang bisa ada di dalam satu dari *state* yang terbatas dalam suatu waktu. Mesin Abstrak tersebut dapat berubah dari suatu *state* ke *state* lainnya berdasarkan suatu *input*, perubahan ini disebut juga dengan transisi, dan suatu fungsi yang me-*mapping* perubahan *state* tersebut, disebut juga dengan fungsi transisi. Sebuah FA didefinisikan oleh *list* dari *state*-nya, *state* awalnya, dan *input* yang membuat transisinya. FA juga memiliki dua tipe, yaitu Deterministik dan Nondeterministik.

Finite Automata mempunyai kapabilitas komputasi yang lebih rendah daripada model komputasi lainnya seperti *Turing Machine*, sehingga ada hal yang tidak bisa dilakukan oleh FA namun bisa dilakukan oleh *Turing Machine*, hal ini terjadi karena memori dari FA terbatas oleh jumlah *state* yang dimilikinya. Jika dibandingkan, kapabilitas komputasi dari *Finite Automata* akan menyerupai dari sebuah *Mesin Turing* yang operasi di *Head* nya dibatasi dengan *read* dan hanya bisa bergerak ke kanan saja. FA bisa direpresentasikan dengan banyak representasi, seperti tabel *state*/transisi, *UML State Machine*, FA Diagram, dan lain-lain.

FA jika dimodelkan dalam model matematika adalah sebagai berikut:

$$FA = (\Sigma, S, s_0, \delta, F)$$

Dimana FA adalah *Finite Automata*, Σ adalah alfabet *input*-nya, S adalah himpunan *state* yang terbatas dan tidak kosong, s_0 adalah *state* awal, yang juga adalah *member* dari S, δ adalah fungsi transisinya, dimana $\delta : S \times \Sigma \rightarrow S$. Terakhir, F adalah himpunan dari *Final State*.

FA sendiri bisa dioptimasi, yang berarti mendapatkan sebuah FA yang memiliki fungsi yang persis sama dengan FA yang ingin dioptimasi dengan *state* yang lebih sedikit. Metode-metode yang bisa digunakan untuk mengoptimasi sebuah FA adalah Algoritma Minimisasi Hopcroft, Tabel Implikasi, dan juga Prosedur Pereduksian Moore.

B. Teori Dasar CFG

Context-Free Grammar atau CFG adalah sebuah *Grammar* Formal yang memiliki aturan produksi dengan bentuk:

$$A \rightarrow \alpha$$

Dengan A adalah simbol *nonterminal* atau *variable* dan α adalah *string* dari *terminal* dan/atau *nonterminal*. Sebuah *Grammar* Formal disebut "*context-free*" jika aturan produksinya bisa diaplikasikan tanpa konteks dari *nonterminal*, sehingga walaupun dikelilingi oleh simbol-simbol lainnya, *Nonterminal* pada bagian kiri akan selalu bisa digantikan oleh bagian kanan atau bagian yang ditunjuk oleh panah.

Sebuah *Grammar* Formal pada intinya adalah sebuah himpunan dari aturan-aturan produksi yang mendeskripsikan semua *string* yang mungkin dalam sebuah bahasa formal. Aturan Produksi adalah sebuah aturan substitusi sederhana, yang bisa diaplikasikan tanpa konteks. Hasil dari sebuah CFG disebut juga dengan CFL atau *Context-free Language*, dimana dua CFG yang berbeda dapat menghasilkan satu CFL yang sama, sehingga penting untuk membandingkan sifat dari bahasanya, yang disebut juga sebagai sifat intrinsik, dan sifat dari *grammar*-nya, yang disebut juga sebagai sifat ekstrinsik, tetapi kedua hal tersebut tetap tidak bisa menyatakan kesamaan dari dua buah bahasa atau *language equality question*.

CFG secara formal didefinisikan sebagai G yang terdiri dari 4-tuple:

$$G = (V, \Sigma, R, S)$$

Dimana V adalah himpunan terbatas dimana semua elemen-nya adalah *nonterminal* atau variabel, Σ adalah himpunan terbatas dari *terminals*, yang tidak boleh berada pada V, dimana *terminals* adalah alfabet dari Bahasa yang didefinisikan oleh *Grammar* G, R adalah relasi terbatas didalam $V \times (V \cup \Sigma) *$, dimana * adalah *Kleene star operation*, dan semua elemen dari R disebut juga aturan produksi dari Grammar G. Terakhir, S adalah Simbol Inisial yang digunakan untuk merepresentasikan kalimat, dan harus merupakan elemen dari Himpunan V.

Parsing & Ambiguitas

Parsing atau derivasi dari sebuah *string* untuk sebuah *grammar* adalah sebuah urutan pengaplikasian aturan dari *grammar* tersebut yang mentransformasikan Simbol *Start* menjadi *string* yang dituju. Hal ini dilakukan untuk membuktikan bahwa sebuah *string* termasuk dalam sebuah *grammar*. Derivasi dapat dilakukan dengan menyebutkan aturan apa yang dipakai dalam suatu Langkah, dan pada simbol yang mana. Derivasi juga bisa dilakukan dengan sebuah diagram *tree* yang disebut *parse tree*, dengan aturan derivasi *leftmost derivation* dan *rightmost derivation*.

Suatu *grammar* juga dapat dibuktikan tidak ambigu dengan cara membuktikan bahwa untuk suatu *string* yang diterima dalam *grammar* tersebut, tidak ada derivasi lain yang memungkinkan.

CNF (Chomsky Normal Form)

CFG juga memiliki bentuk-bentuk baku seperti BNF yaitu *Backus-Naur Form* dan juga CNF yaitu *Chomsky Normal Form*. Bentuk baku yang digunakan untuk CFG pada Tugas ini adalah bentuk CNF, karena dibutuhkan dalam algoritma CYK (*Cocke-Younger-Kasami*) untuk mengecek apakah suatu input masukan termasuk dalam sebuah grammar atau tidak. CNF sendiri terdiri dari hanya 2 Aturan Produksi:

$$S \rightarrow a$$

$$S \rightarrow AB$$

atau didefinisikan sebagai

1. Tidak memiliki aturan produksi yang tidak berguna

2. Tidak memiliki aturan produksi *unit*
3. Tidak memiliki aturan produksi ϵ

Sehingga dalam suatu CNF hanya diperbolehkan untuk mempunyai campuran dari kedua Aturan diatas, yang berarti bisa saja ada gabungan antara 1 *terminal* atau 2 *nonterminal*:

$$S \rightarrow a \mid AB \mid b$$

$$A \rightarrow a$$

$$B \rightarrow b$$

Untuk mengkonversikan suatu CFG ke bentuk CNF, dilakukan algoritma sebagai berikut:

1. Acuhkan aturan produksi yang terkonversi dalam bentuk CNF
2. Substitusi aturan produksi yang bagian kanannya mengandung *terminal* menjadi suatu aturan produksi baru untuk setiap *terminal* yang tergabung dalam *nonterminal* lainnya, contoh: AcBd menjadi ACBD, C -> c, D -> d
3. Substitusi aturan produksi yang bagian kanannya memuat lebih 2 *nonterminal*, contoh: ABD menjadi AC, C -> BD
4. Lakukan langkah nomor 2 dan 3 terus menerut sampai akhirnya terbentuk CNF
5. Aturan produksi dan juga *nonterminal* mungkin saja didapatkan dari hasil konversi ke CNF tersebut

Algoritma CYK (Cocke-Younger-Kasami)

Algoritma CYK atau *Cocke-Younger-Kasami* adalah sebuah algoritma *parsing* untuk CFG yang digunakan untuk mengecek apakah suatu *string* diterima atau tidak oleh CFG tersebut. Ditemukan oleh Itiroo Sakai pada tahun 1961, Algoritma CYK ditemukan ulang oleh John Cocke, Daniel Younger, dan Tadao Kasami, yang mengimplementasikan Algoritma CYK dengan *bottom-up parsing* dan *dynamic programming*. Algoritma CYK hanya dapat digunakan untuk CFG yang sudah dikonversi menjadi CNF, sehingga diperlukan suatu implementasi juga untuk mengkonversi suatu CFG sembarang menjadi bentuk CNF-nya. Pentingnya algoritma CYK terdapat dari efisiensinya yang tinggi, dengan notasi *Big-O* sebesar $O(n^3 * |G|)$ dimana n adalah panjang *string* yang di *parsing* dan |G| adalah ukuran dari *grammar* CNF G. Algoritma-nya dalam *pseudocode* adalah sebagai berikut:

let the input be a string *I* consisting of *n* characters: $a_1 \ldots a_n$.
let the grammar contain *r* nonterminal symbols $R_1 \ldots R_r$, with start symbol R_1.
let *P*[*n*,*n*,*r*] be an array of booleans. Initialize all elements of *P* to false.
let *back*[*n*,*n*,*r*] be an array of lists of backpointing triples. Initialize all elements of *back* to the empty list.

for each *s* = 1 to *n*
  for each unit production $R_v \rightarrow a_s$
    set *P*[*1*,*s*,*v*] = true

```
for each l = 2 to n -- Length of span
   for each s = 1 to n-l+1 -- Start of span
      for each p = 1 to l-1 -- Partition of span
         for each production Ra    → Rb Rc
            if P[p,s,b] and P[l-p,s+p,c] then
               set P[l,s,a] = true,
               append <p,b,c> to back[l,s,a]

if P[n,1,1] is true then
   I is member of language
   return back -- by retracing the steps through back, one can easily construct all
possible parse trees of the string.
else
   return "not a member of language"
```

Pseudocode Algoritma CYK
sumber dari Wikipedia.org

C. Penjelasan Syntax JavaScript yang Perlu Diperhatikan dalam Pembuatan CFG dan FA

JavaScript adalah inti dari banyak teknologi dari *World Wibe Web*, dimana banyak website menggunakan *JavaScript* untuk *client side* dari *webpage behaviour*, biasanya juga mengimplementasikan *third party libraries*. *JavaScript* sendiri memiliki fitur yaitu tidak adanya definisi tipe dari data, atau biasa disebut juga *weakly typed*, sehingga banyak dari operasi-operasi data dari tipe yang berbeda-beda bisa tergabung. *JavaScript* juga adalah bahasa pemrograman dengan *multi-paradigm*, yang mendukung tipe pemrograman *event-driven, functional*, dan *imperative*. Hal-hal dalam sintaks JavaScript yang perlu diperhatikan saat membuat CFG dan FA nya adalah sebagai berikut:

1. Whitespace dan Space

Beberapa *syntax* di JavaScript perlu memerhatikan penggunaan *Whitespace* dan *space*, dimana setelah *for* bisa diisi *whitespace* sebanyak apapun bahkan tidak ada spasi pun bisa, tetapi harus ada separator minimal 1 spasi diantara var dan I.

Misal : for (var I = 0 ; I < 5 ; I++) { \n \n }

Sehingga pada kelompok kami, kami melakukan *preprocessing* untuk semua *whitespace* (kecuali untuk yang membutuhkan minimal 1 *space* seperti var dan I, akan disisakan 1).

2. Pemanggilan return atau break didalam function atau loop

Pemanggilan *return* didalam sebuah *function* atau *break/continue* didalam sebuah *loop* juga perlu diperhatikan, karena jika ada sebuah *sub-block-of-code* di dalam *function* atau *loop*,

maka pemanggilan *return* dan *break/continue* juga masih bisa dilakukan. Sehingga perlu disesuaikan dengan implementasi yang ada.

3. End of Statement diakhiri oleh *semicolon*

Dalam *JavaScript*, akhir dari sebuah *statement* diakhiri oleh *semicolon* atau *newline*, sehingga dalam implementasi kami, kami menggunakan *semicolon* untuk *End Of Statement*.

4. For Loop

For Loop didalam *JavaScript* menggunakan format yang serupa dengan C, yaitu

```
for (var i = 0; i < 5; i++) {
  /* Block of code */
}
```

Dimana diharuskan menggunakan 2 buah separator *semicolon* didalam *bracket* dan diperbolehkan hampir semua sintaks ditambah dengan break/continue.

5. Nama variabel

Dalam *JavaScript*, tidak diperbolehkan menggunakan nama variabel yang diawali dengan angka atau simbol. Sehingga hal tersebut perlu dicek oleh sebuah FA.

6. Ekspresi Matematis

JavaScript menggunakan Ekspresi Matematis dengan bentuk var atau angka operasi var atau angka, dimana hal ini juga merangkup string didalamnya. Angka hanya boleh diisi dengan angka, dan string isinya dibebaskan kecuali *newline*, diantara kedua tanda petik “, tetapi *newline* diperbolehkan diantara kedua tanda petik ‘.

7. Komen dan dokumentasi

Komen pada *JavaScript* ada dua jenis, yaitu *single line* dan *multi-line*, dimana *comment* didalam *single line* dan *multi-line* dibebaskan isinya, tetapi tidak diperbolehkan *newline* pada *single line comment*.

8. Assignment

Assignment pada *JavaScript* menggunakan 2 alternatif, yaitu var/let/const lalu variabel = variabel atau Ekspresi Matematis (termasuk *string*, *number*, dan lain-lain), dan juga variabel = variabel atau Ekspresi Matematis. Tetapi juga bisa dilakukan += -= dst. untuk variabel.

9. Call Method atau Attribute

Dalam *JavaScript*, dapat memanggil *Method* atau *Attribute* dari sebuah Objek, sehingga diperlukan implementasi khusus untuk pengecekan pemanggilan tersebut, tetapi karena semantik tidak dipermasalahkan, maka pemanggilan *Method* atau *Attribute* tidak harus dari *Method* atau *Attribute* yang terdefinisi.

BAB II
Hasil FA dan CFG

A. Hasil Finite Automata

Finite Automata yang kami rancang dibagia menjadi beberapa bagian, ada yang berbentuk REGEX, ada yang berbentuk DFA. Tetapi semua Regex sudah merepresentasikan DFA dan semua DFA juga sudah dapat menggantikan Regex.

Regex

Regex	Penjelasan
\/*[\w\W]*?*\/	Multiline comment
\/\/.*	Comment
([0-9]+(\.[0-9]+)+)|([0-9]+\.)|(\.[0-9]+)|([0-9]+(\.[0-9]+)*)	Integer dan float
(?:[^a-zA-Z_$0-9]|^)((?:true)|(?:false))(?:[^a-zA-Z_$0-9]\|$)	True and false
(?:^|[^a-zA-Z_$0-9])([a-zA-Z_$0-9]+)	Variable
[0-9][^0-9]+	Invalid variable
((?:return)|(?:var)|(?:let)|(?:const)|(?:delete)|(?:function)| (?:case))(?:\s+)((?:[({\[]*\s*variable)|(?:[({\[]*\s*number))	Mencari spasi yang dibutuhkan supaya tidak dihapus.

DFA

fa_var.py	
State = {Start, var} Alphabet = Ascii character Start State = {Start} Accept State = {var}	Transition function: `{'Start': {'DUL': 'var' },` `'var': {'number': 'var', 'DUL': 'var'}}`
fa_mathexp.py	
State = {Start, var, literal, literal_with_point_only, literal_with_point, after_obj, wait_power, wait_equal, wait_strict_equal, wait_gt, wait_rshift, wait_lt, wait_logical_andm wait_logical_or, after_between_ops, after_hugging_operator} Alphabet = Ascii character Start State = {Start} Accept State = {literal, literal_with_point, var, after_obj}	Transition function: `{'Start': {' ': 'Start', 'DUL': 'var', 'number': 'literal', '.': 'literal_with_point_only', '+': 'after_hugging_operator', '-': 'after_hugging_operator', '~': 'after_hugging_operator', '!': 'after_hugging_operator'},` `'var': {' ': 'after_obj', 'number': 'var', 'DUL': 'var', 'between_ops': 'after_between_ops', '+': 'after_between_ops', '-': 'after_between_ops', '=': 'wait_equal', '!': 'wait_equal', '>': 'wait_gt', '<': 'wait_lt', '*': 'wait_power', '|': 'wait_logical_or', '&': 'wait_logical_and'},` `'literal': {' ': 'after_obj', 'number': 'literal', '.':`

```
'literal_with_point', 'between_ops':
'after_between_ops', '+': 'after_between_ops', '-
': 'after_between_ops', '=': 'wait_equal', '!':
'wait_equal', '>': 'wait_gt', '<': 'wait_lt', '*':
'wait_power', '|': 'wait_logical_or', '&':
'wait_logical_and'},
                        'literal_with_point_only':
{'number': 'literal_with_point'},
                        'literal_with_point': {'
': 'after_obj', 'number': 'literal_with_point',
'between_ops': 'after_between_ops', '+':
'after_between_ops', '-': 'after_between_ops',
'=': 'wait_equal', '!': 'wait_equal', '>':
'wait_gt', '<': 'wait_lt', '*': 'wait_power', '|':
'wait_logical_or', '&': 'wait_logical_and'},
                        'after_obj': {' ':
'after_obj', 'between_ops': 'after_between_ops',
'+': 'after_between_ops', '-':
'after_between_ops', '=': 'wait_equal', '!':
'wait_equal', '>': 'wait_gt', '<': 'wait_lt', '*':
'wait_power', '|': 'wait_logical_or', '&':
'wait_logical_and'},
                        'wait_power': {'*':
'after_between_ops', ' ': 'after_between_ops',
'~': 'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_equal': {'=':
'wait_strict_equal'},
                        'wait_strict_equal': {'=':
'after_between_ops', ' ': 'after_between_ops',
'~': 'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_gt': {'>':
'wait_rshift', ' ': 'after_between_ops', '=':
'after_between_ops', '~':
'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
```

```
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_rshift': {'>':
'after_between_ops', ' ': 'after_between_ops',
'~': 'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_lt': {'<':
'after_between_ops', ' ': 'after_between_ops',
'=': 'after_between_ops', '~':
'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_logical_and': {'&':
'after_between_ops', ' ': 'after_between_ops',
'~': 'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'wait_logical_or': {'|':
'after_between_ops', ' ': 'after_between_ops',
'~': 'after_hugging_operator', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal'},
                        'after_between_ops': {' ':
'after_between_ops', 'DUL': 'var', 'number':
'literal', '.': 'literal_with_point_only', '+':
'after_hugging_operator', '-':
'after_hugging_operator', '~':
'after_hugging_operator', '!':
'after_hugging_operator'},
                        'after_hugging_operator':
{' ': 'after_hugging_operator', '!':
'after_hugging_operator', 'DUL': 'var', 'number':
'literal', '.': 'literal_with_point_only'},
                        }
```

B. Hasil Context-Free Grammar

Context-Free Grammar yang kami rancang untuk bahasa pemrograman JavaScript dapat dibagi menjadi beberapa bagian, dimana ada CFG untuk bagian utama, dan beberapa bagian tambahan untuk membantu CFG utama. CFG utama yang kami rancang adalah sebagai berikut:

CFG

Terminals:
variable number = ! < > \(\) { } \[ ] \. ; , : \+ * % - \/ & \^ \? \| break default for return var const delete function switch while case else if throw catch let try continue finally

Variables:
START OPTSTART ARGUMENT ARRAYVAR ASSIGNMENT ASSIGNOPS BREAKCONTINUE CASE CASEDEFAULT CATCH DEFAULT DELETE ELSE FINALLY FOR FUNCTION IF MATHEXP NUMBER OBJECT OPERATORS PARAM PARAMREST RETURN SEMICOLON SPACE SWITCH THROW TRY VAR VARLETCONST WHILE

Productions:
START -> START START | ASSIGNMENT | FOR | IF | WHILE | FUNCTION | SPACE | BREAKCONTINUE | RETURN | SWITCH
OPTSTART -> START | e
ARGUMENT -> ARGUMENT , ARGUMENT | . . . MATHEXP | MATHEXP | e
ASSIGNMENT -> VARLETCONST SPACE VAR = MATHEXP SEMICOLON | VAR ASSIGNOPS MATHEXP SEMICOLON | VARLETCONST SPACE VAR SEMICOLON | MATHEXP SEMICOLON | VARLETCONST SPACE { PARAMREST } = MATHEXP SEMICOLON | { PARAMREST } = MATHEXP SEMICOLON | VARLETCONST SPACE [PARAMREST] = MATHEXP SEMICOLON | [PARAMREST] = MATHEXP SEMICOLON
ASSIGNOPS -> = | + = | - = | * = | / = | * * = | % = | & & = | & = | or or = | or = | ^ = | ? ? = | = < < | = > > | = > > >
BREAKCONTINUE -> break SEMICOLON | continue SEMICOLON
CASE -> case SPACE MATHEXP : { OPTSTART }
CASEDEFAULT -> CASEDEFAULT CASEDEFAULT | CASE | DEFAULT | e
CATCH -> catch (variable) { OPTSTART }
DEFAULT -> default : { OPTSTART }
DELETE -> delete SPACE MATHEXP SEMICOLON
ELSE -> else { START } | else SPACE IF | e
FINALLY -> finally { OPTSTART }
FOR -> for (VARLETCONST SPACE VAR = MATHEXP SEMICOLON MATHEXP SEMICOLON MATHEXP) { OPTSTART } | for (VAR = MATHEXP SEMICOLON MATHEXP SEMICOLON MATHEXP) { OPTSTART }
FUNCTION -> function SPACE variable (PARAMREST) { OPTSTART } | function SPACE variable ( ) { OPTSTART } braces
IF -> if (MATHEXP) { START } ELSE

MATHEXP -> ! MATHEXP | (MATHEXP) | VAR | NUMBER | MATHEXP OPERATORS MATHEXP | VAR + + | VAR - - | [ARGUMENT] | { OBJECT }
NUMBER -> number
OBJECT -> OBJECT , OBJECT | number : MATHEXP | variable : MATHEXP | variable
OPERATORS -> + | * | * * | - | / | % | & & | & | or or | or | ^ | = = | = = = | ! = = | ! = | < | > | < = | > = | ? ? | < < | > > | > > >
OPTSTART -> START | e
PARAM -> PARAM , PARAM | variable
PARAMREST -> PARAM | PARAM , . . . variable | e
RETURN -> return SPACE MATHEXP SEMICOLON | return SEMICOLON | e
SEMICOLON -> ;
SPACE -> SPACE SPACE | space | newline
SWITCH -> switch (MATHEXP) { CASEDEFAULT }
THROW -> throw SPACE MATHEXP SEMICOLON
TRY -> try { OPTSTART } CATCH FINALLY | try { OPTSTART } CATCH | try { OPTSTART } FINALLY
VAR -> variable | VAR (ARGUMENT) | VAR . VAR | (VAR) | VAR [MATHEXP]
VARLETCONST -> var | let | const
WHILE -> while (MATHEXP) { OPTSTART }

Selanjutnya, untuk CFG tambahan yang kami gunakan adalah untuk mengecek apakah return berada didalam fungsi, dan apakah break ada di dalam loop atau switch case, berikut adalah CFG nya:

CFG Untuk Function

Terminals: function ( ) { } return e

Variables: START FUNCTION BRACES FUNCINSIDE

Productions:
START -> START START | FUNCTION | BRACES
FUNCTION -> function ( ) { FUNCINSIDE }
FUNCINSIDE -> { FUNCINSIDE } | (FUNCINSIDE) | BRACES | FUNCINSIDE FUNCINSIDE | return | START
BRACES -> { START } | (START) | { } | ( ) | BRACES BRACES | e

CFG Untuk Loop dan Switch

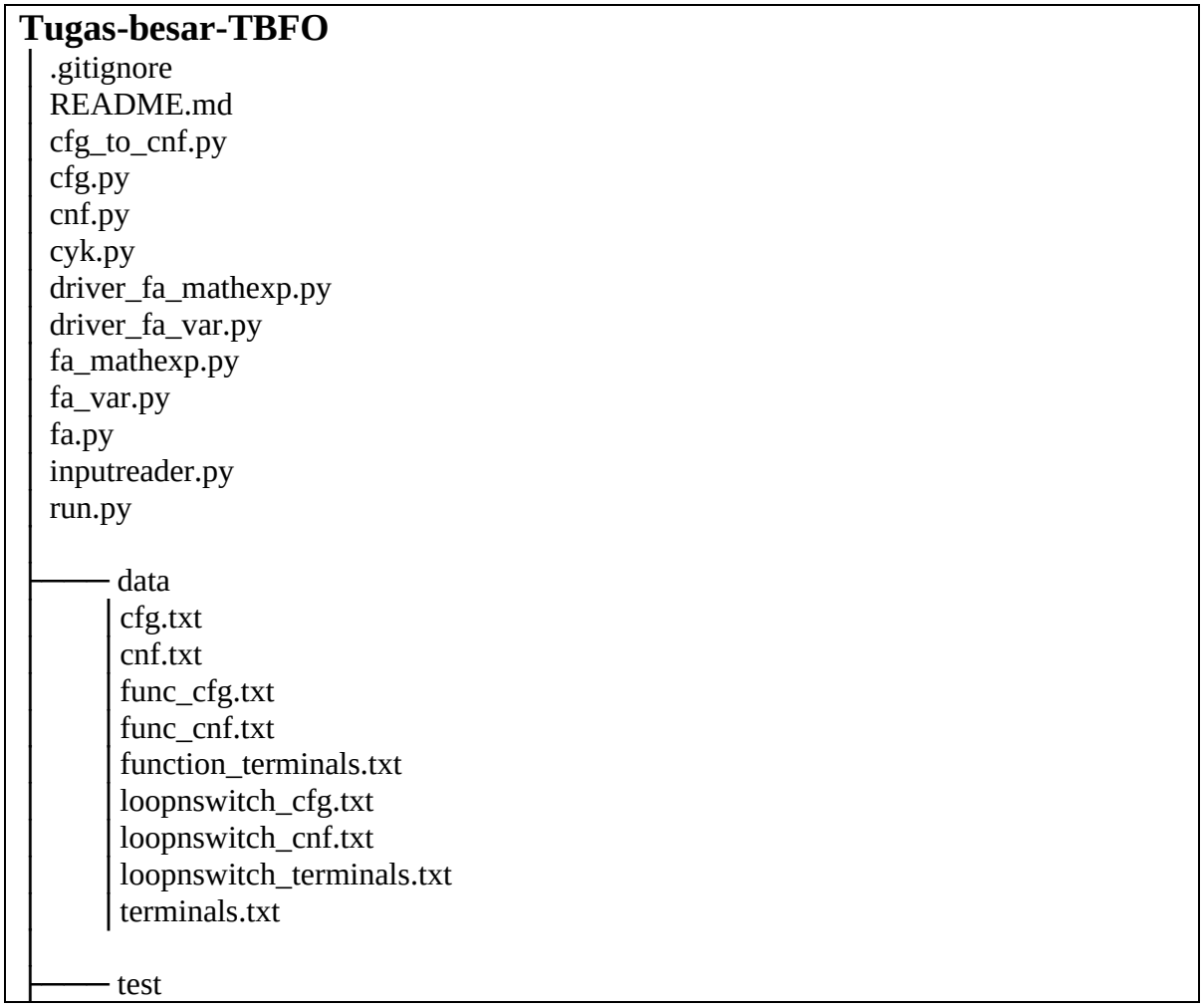
Terminals: ( ) { } for switch while break continue

Variables: START LNS BRACES LNSINSIDE

Productions:
START -> START START | LNS | BRACES
LNS -> while ( ) { LNSINSIDE } | for ( ) { LNSINSIDE } | switch ( ) { LNSINSIDE }
LNSINSIDE -> { LNSINSIDE } | (LNSINSIDE) | BRACES | LNSINSIDE LNSINSIDE | break | continue | START
BRACES -> { START } | (START) | { } | ( ) | BRACES BRACES | e

BAB III
Implementasi dan Pengujian

A. Struktur Data



B. Fungsi dan Prosedur

3.1 File cfg_to_cnf.py

File ini digunakan untuk langsung dijalankan (dengan command python cfg_to_cnf.py) yang akan memproses CFG dan mengkonversikan dan menyimpan CNF untuk CFG utama, CFG fungsi, dan CFG loop serta switch.

3.2 File cfg.py

File ini berisi class CFG yang berfungsi untuk membaca, memproses, dan mengubah CFG ke CNF.

class CFG

Header	Penjelasan
`def __init__(self) -> None:`	Konstruktor class CFG, inisiasi atribut rules dengan dictionary kosong
`def load(self, filename: str) -> None:`	Membaca dan memproses file berisi production rules CFG
`def to_cnf(self) -> CNF:`	Mengubah CFG ke CNF dengan mengembalikan object ckass CNF

3.3 File cnf.py

File ini berisi class CNF yang berfungsi untuk membaca dan menyimpan rules CNF.

class CNF

Header	Penjelasan
`def __init__(self):`	Konstruktor class CNF, inisiasi atribut rules dengan dictionary kosong dan start symbol dengan “START”.
`@staticmethod` `def get_var_name(i: int) -> str:`	Static method untuk mendapatkan nama variabel berdasarkan urutannya. (1 untuk A, 2 untuk B, 27 untuk A1, dst.)
`def dump(self, filename: str):`	Menyimpan rules pada object CNF ke dalam file.
`def load(self, filename: str):`	Membaca dan memproses file berisi production rules CNF.
`def subtitute_all_postfix(self, postfix, replacement: str):`	Digunakan dalam konversi CFG ke CNF untuk mengubah dua symbol menjadi satu symbol. (ABC menjadi AD)
`def subtitute_all(self, old: str, new: str):`	Digunakan dalam konversi CFG ke CNF untuk mengubah suatu rule menjadi rule yang lainnya. (a menjadi A)

3.4 cyk.py

File ini berisi class CYK yang berfungsi untuk memuat CNF dan menjalankan algoritma CYK.

class CYK

Header	Penjelasan
`def __init__(self, filename):`	Konstruktor class CYK, untuk memuat CNF sesuai dengan filename. (Akan memanggil fungsi load_cnf)
`def load_cnf(self, filename):`	Membuat class CNF kemudian memanggil fungsi load lalu menyimpan rules dan start symbol pada CYK.
`def check(self, words):`	Menjalankan algoritma CYK sesuai dengan rules dan start symbol yang telah disimpan. Parameter words berisi list of terminal yang akan dicek.

3.5 fa.py

File ini berisi fungsi-fungsi untuk memproses string kode supaya bisa diproses di CYK.

Header	Penjelasan
`class SyntaxError(Exception):`	Class untuk handle syntax error.
`def get_current_line(code: str, idx: int):`	Untuk mendapatkan baris kode yang sesuai berdasarkan indeks pada string, digunakan untuk mencari baris kode program yang error.
`def process_string(code: str):`	Memproses string pada kode yang akan dicek dan mengubahnya menjadi terminal "number".
`def tokenize_with_fa(code: str, terminals: list[str]):`	Melakukan proses pada string kode (bagian comment, string, angka, boolean, dan variable) sehingga bisa diproses di CYK.

3.6 inputreader.py

File ini berisi fungsi-fungsi yang digunakan untuk memproses string kode menjadi list of terminal yang akan diproses CYK.

Header	Penjelasan
`def preprocessing(w: str):`	Melakukan preprocessing pada string kode yaitu menghilangkan whitespace yang tidak diperlukan untuk mengoptimasi CYK.
`def inputread(w: str, terminals: list):`	Memproses string kode menjadi list of terminal yang akan diproses CYK.

3.7 run.py

File ini digunakan sebagai entry point program utama, yaitu memproses file atau folder yang berisi test case string kode yang akan di-parse. Dijalankan dengan command "python run.py <nama_file_atau_folder>. Jika input berupa file maka hanya akan memproses satu file tersebut, sedangkan jika input berupa folder, akan memproses seluruh file yang ada dalam folder tersebut.

Header	Penjelasan
`def print_duration(duration):`	Untuk mencetak durasi parsing.
`def check_file(filename: str, count=None):`	Untuk melakukan proses parsing pada file tertentu.
`def success(message):`	Untuk menampilkan pesan parsing sukses.
`def fail(message):`	Untuk menampilkan pesan parsing error.

`def function_check(w: str):`	Untuk mengecek apakah fungsi-fungsi pada string valid, termasuk mengecek return di luar function (error).
`def loopnswitch_check(w: str):`	Untuk mengecek apakah break berada pada loop ataupun switch yang valid

3.8 fa_var.py

Header	Penjelasan
`def isletter(c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan huruf alfabet
`def isNumber(c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan angka numerik
`def isDollarUnderscoreLetter (c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan tanda dollar, underscore, ataupun huruf alfabet
`def check_fa_var(string: str) -> bool:`	Untuk mengecek dengan fa apakah suatu string merupakan nama variabel yang valid

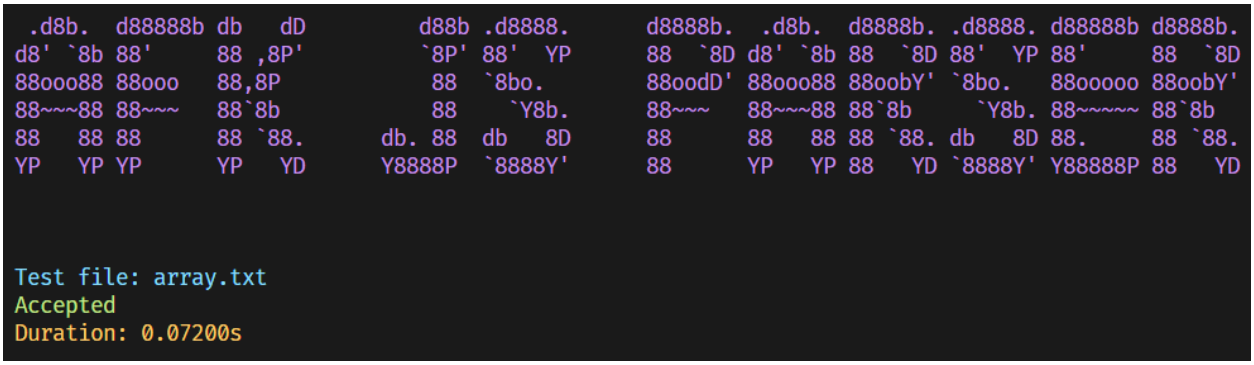
3.9 fa_mathexp

Header	Penjelasan
`def isletter(c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan huruf alfabet
`def isNumber(c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan angka numerik
`def isDollarUnderscoreLetter (c: str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan tanda dollar, underscore, ataupun huruf alfabet
`def is_between_ops(c : str) -> bool:`	Untuk mengecek apakah suatu karakter merupakan karakter ‘/’, ‘%’, ‘^’
`def check_mathexp(string: str) -> bool:`	Untuk mengecek dengan fa apakah suatu string merupakan math expression yang valid

C. Antarmuka

Antarmuka yang kelompok kami buat berbasis command-line, dengan terminal mengeluarkan hasil dari pengecekan suatu *input text*, apakah diterima atau tidak.

Berikut adalah *screenshot* dari Antarmuka yang kelompok kami gunakan



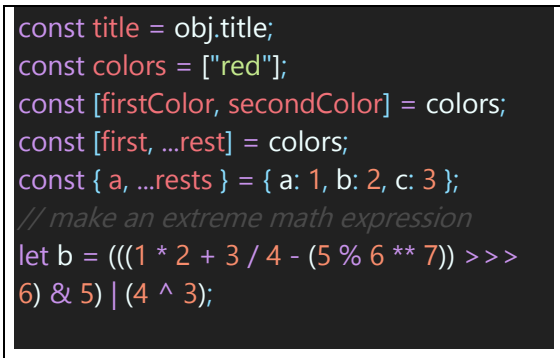
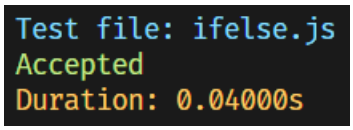
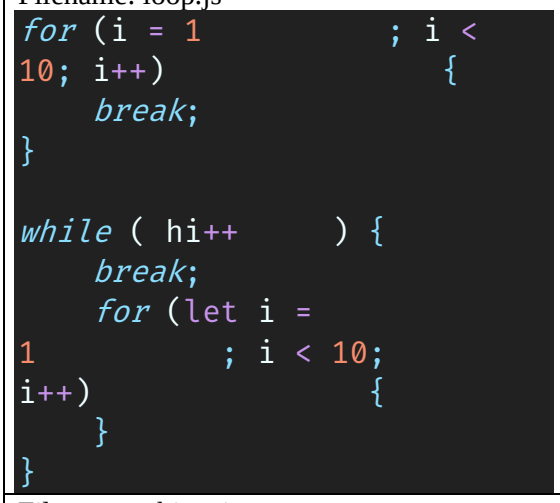
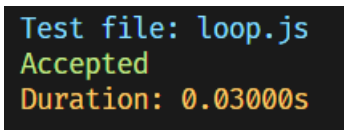
Gambar 3.1 Screenshot dari Antarmuka

D. Kasus Uji dan Analisis Hasil

3.1 Uji CFG

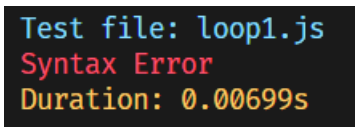
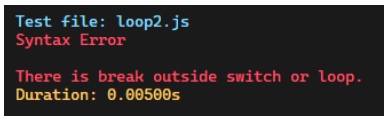
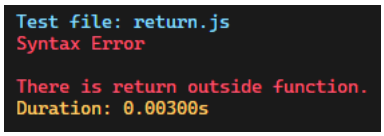
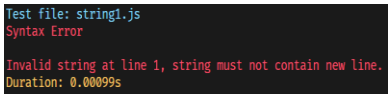
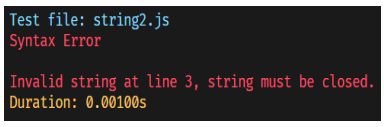
Input	Hasil	Penjelasan
Filename: a.js `let a = input();` `if (a == 0) {` `  console.log("zero haish");` `} else if (a < 5) {` `  console.log("more more more");` `} else {` `  console.log("YASHHH");` `}`	`Test file: a.txt` `Accepted` `Duration: 0.04300s`	Secara sintaks, kasus if, console.log, else, dan persamaan Boolean tidak menyalahi aturan, sehingga tidak ada *error*
Filename: array.js `const array = [1, 2, 3, 4];` `for (let i = 0; i < 4; i++)` `{` `  array.push(array[i] * 2);` `  console.log(array[i]);` `}` `console.log(array.length);`	`Test file: array.txt` `Accepted` `Duration: 0.07200s`	Secara sintaks, deklarasi array, loop, dan pengaksesan array.length tidak menyalahi aturan

`console.log(array[array.length - 1]);`		
Filename: break1.js `let i = 0;` `while (i < 6) {` `  if (i === 3) {` `  break;` `  }` `  i = i + 1;` `}` `console.log(i);` `// expected output: 3`	`Test file: break1.js` `Accepted` `Duration: 0.03300s`	Secara sintaks, break sudah ditempatkan dengan benar yaitu didalam loop
Filename: function.js `function add(a, b) {` `  return a + b;` `}` `function subtract(a, b) {` `  return a - b;` `}` `function multiply(a, b) {` `  return a * b;` `}` `function divide(a, b) {` `  return a / b;` `}`	`Test file: function.js` `Accepted` `Duration: 0.05400s`	Secara sintaks, seluruh *testcase function* sudah benar
Filename: const.js `const date1 = Date("December 17, 1995 03:24:00");` `// Sun Dec 17 1995 03:24:00 GMT...` `const date2 = Date("1995-12-17T03:24:00");` `// Sun Dec 17 1995 03:24:00 GMT...` `const { bar, foo } = 1;` `const { name } = obj;` `const { age, id } = obj;` `const email = obj.email;`	`Test file: const.js` `Accepted` `Duration: 0.26899s`	Secara sintaks, penggunaan *const* sudah tepat.

		
Filename: ifelse.js 		Secara sintaks, penggunaan *if-else*, dan *else if* sudah benar
Filename: loop.js 		Secara sintaks, *whitespace* didalam () for dan while diperbolehkan, sehingga tidak menyalahi aturan
Filename: object.js 		Secara sintaks, deklarasi object didalam JavaScript

<pre> a: 5, b: 8**2, c: whatever, 5: yooo, nested: { shouldbevalid: yes } }; obj.additional = "ini object"; object.nested.shouldbevalid = update;</pre>	<pre>Test file: object.js Accepted Duration: 0.03500s</pre>	seperti ini diperbolehkan, dan cara pengaksesan *attribute* juga tidak menyalahi aturan
Filename: random1.js <pre>const as = "Hello world!"; if (as.length < 5) { console.log("lol"); } let bcd = `  anjay ini panjang bet coy   `; let a = false; function aenjeaye(satu, dua, tiga) { let count = 1; while (true) { count++; if (count > 100) { break; } } switch (a) { case 1: { console.log("maap"); } default: { count++;</pre>	<pre>Test file: random1.js Accepted Duration: 0.13701s</pre>	Secara sintaks, tidak ada yang menyalahi aturan dari *testcase* tersebut

`` } } return count; } for (let i = 1; i < 5; i++) { console.log(i); } ```		
Filename: switch1.js ``` switch (expr) { case 1: { var x = 5; break; } case 2: { let x = 4; break; } default: { const x = 3; break; } } ```	``` Test file: switch1.js Accepted Duration: 0.02200s ```	Secara sintaks, tidak ada yang menyalahi aturan dari *statement switch-case* tersebut
Filename: const1.js ``` if (true) const a = 1; // SyntaxError: Unexpected token 'const' ```	``` Test file: const1.js Syntax Error Duration: 0.00599s ```	Secara sintaks, *error* karena ada deklarasi const pada tempat yang salah
Filename: ifelse1.js ``` if () { let a = 5; } ```	``` Test file: ifelse1.js Syntax Error Duration: 0.00700s ```	Secara sintaks *error* karena tidak ada ekspresi didalam *if*
Filename: ifelse2.js ``` else { if (x) { var x = 1; } } ```	``` Test file: ifelse2.js Syntax Error Duration: 0.00800s ```	Secara sintaks *error* karena ada *else* diluar tanpa *if*
Filename: ifelse3.js ``` else if { // ini salah woy } ```	``` Test file: ifelse3.js Syntax Error Duration: 0.00500s ```	Secara sintaks *error* karena ada *else* diluar tanpa *if*

Filename: loop1.js `for(let i = 5 i < 10; i++) {` `}`		Secara sintaks *error* karena kurang *semicolon* diantara *parameter* pertama dan kedua
Filename: loop2.js `let b = 10298109;` `// ini di luar loop kok ada break` `break;`		Secara sintaks *error* karena ada *break* diluar *loop* atau *switch*. Program kami juga mengimplementasikan '*bonus'* untuk memperlihatkan letak kesalahan, yaitu pada *loop* atau *switch*
Filename: return.js `return;` `// ini harusnya ke reject`		Secara sintaks *error* karena terdapat *return* diluar *function*. Program kami mengimplementasikan '*bonus*' untuk memperlihatkan letak kesalahan, yaitu pada *function*
Filename: string1.js `"hiya` `"`		Secara sintaks *error* karena terdapat *newline* pada *one-line string*, atau *unterminated string literal*. Kelompok kami mengimplementasikan bonus untuk memperlihatkan letak kesalahan *invalid string at line X*, dan jenis kesalahannya yaitu *string must not contain new line*
Filename: string2.js `let a = "ini masih valid"` `const s = ` `  `ini juga valid????`		Secara sintaks *error* karena tidak terdapat penutup *string* ` untuk *const s*. Kelompok kami mengimplementasikan bonus untuk memperlihatkan letak kesalahan *invalid string at line X*, dan

		jenis kesalahannya yaitu *string must be closed*

3.2 Uji FA

Uji Expresi Matematika

Input	Hasil	Penjelasan
`1 + a - 2 +-5-+4 * 6 ** 6 / 7 && !1 | var1 <= 3 << 23 >>> tujuh < 10 >= 40 !== 3`	True	Secara sintaks math expression sudah benar
`!! 5 + ! ! ! 4 + tujuh << 4`	True	Secara sintaks math expression sudah benar
`5 ! 4`	Error karena memasukkan pada state wait_equal False	5 ! 4 tidak valid
`5 !== 4`	True	Secara sintaks math expression sudah benar

Uji Penamaan Variabel

Input	Hasil	Penjelasan
`$_12e_`	True	Secara sintaks penaman variabel sudah benar
`1_fakhri`	Error karena memasukkan number pada state Start False	Pada penamaan variabel tidak boleh ada
`var^fa`	Error karena memasukkan ^ pada state var False	Pada penamaan variabel tidak boleh ada karakter selain $, _, huruf alfabet, dan nomor

LAMPIRAN

Link GitHub: Repository

Pembagian Tugas: Workspace

No	TODO	Nama	Progress
1	CFG	Arsa Ken Fakhrit	Done
2	FA	Fakhri	Done
3	CFG → CNF	Arsa	Done
4	CYK + Input Parser	Ken	Done

REFERENSI

https://www.w3schools.com/js/ diakses pada 12 November 2022 pukul 19.05 WIB

https://en.wikipedia.org/wiki/JavaScript diakses pada 12 November 2022 pukul 19.47 WIB

https://en.wikipedia.org/wiki/Finite-state_machine diakses pada 12 November 2022 pukul 20.04 WIB

https://en.wikipedia.org/wiki/Context-free_grammar diakses pada 12 November 2022 pukul 21.47 WIB

https://en.wikipedia.org/wiki/CYK_algorithm diakses pada 12 November 2022 pukul 22.00 WIB