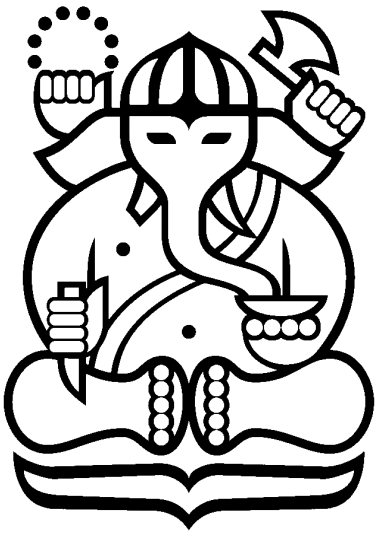
IF2124 Teori Bahasa Formal dan Otomata

PROGRAM PARSER BAHASA PEMROGRAMAN JAVASCRIPT DENGAN CONTEXT-FREE GRAMMAR DAN FINITE AUTOMATA

Laporan Tugas Besar

Disusun untuk memenuhi tugas besar Mata Kuliah Teori Bahasa Formal dan Otomata pada Semester 1 (satu) Tahun Akademik 2022/2023



Disusun oleh

Rinaldy Adin 13521134

Enrique Alifio Ditya 13521142

Rava Maulana Azzikri 13521149

Kelompok Bob

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

BAB I
TEORI DASAR

1. *Finite Automata*

Finite Automata merupakan suatu model mesin abstrak yang terdiri atas tuple dari 5 elemen. *Finite Automata* memiliki himpunan *states* dan *rules* untuk memindahkannya dari satu *state* ke lainnya atas input tertentu. Dengan ini, *finite state automata* menggambarkan pemodelan matematika dari suatu mesin digital yang dapat mengenali suatu pola dari input yang diambil dari suatu himpunan karakter.

Lima elemen tuple yang mendirikan suatu *Finite Automata* adalah himpunan *state* berhingga, state awal, set final state, input, dan aturan perpindahan state. Penguraiannya adalah sebagai berikut.

$$\text{FA} = (\ Q,\ \Sigma,\ q,\ F,\ \delta\ )$$

FA = *Finite Automata,*

Q = Himpunan set berhingga,

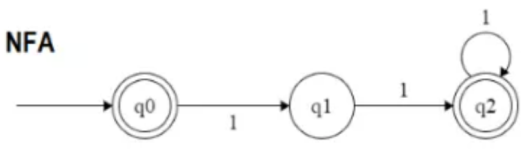
Σ = Himpunan simbol input,

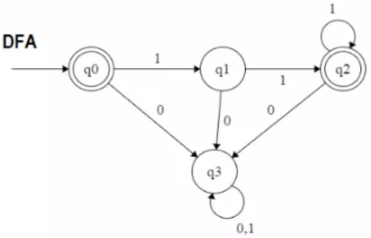
q = State awal,

F = Himpunan state akhir,

δ = Aturan perpindahan state / fungsi transisi.

Finite Automata terbagi menjadi dua jenis, yakni *Deterministic Finite Automata* (DFA) dan *Non-deterministic Finite Automata* (NFA). DFA merupakan subset dari NFA, yakni suatu automata yang tidak dapat menerima *null* input (ε) sehingga tidak dapat berpindah dari satu state ke lainnya tanpa menerima input apapun. NFA pada sisi lain merupakan suatu automata yang dapat menerima *null* dan dapat berpindah ke satu state atau lebih dengan *null* atau satu input. DFA dan NFA memiliki bentuk yang *interchangeable*, yakni bisa dikonversi dari satu bentuk ke lainnya tanpa mengubah bahasa yang diterima dari automata. Berikut adalah contoh dari keduanya.





2. *Context-free Grammar*

Dalam Teori Bahasa Formal, *Context-free Grammar* adalah *formal grammar* yang memiliki produksi dalam bentuk

$$A \rightarrow \alpha$$

dengan *A* sebagai satu simbol non-terminal yakni simbol penghasil produksi dan α sebagai satu simbol terminal berupa hasil produksi. Simbol non-terminal hanyalah berperan dalam proses derivasi string, namun tidak muncul pada string akhir layaknya simbol terminal. Pada dasarnya, *Context-free Grammar* (CFG) merupakan tuple berisi 4 elemen, yakni

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

G = *Grammar,*

V = Himpunan simbol non-terminal berhingga,

Σ = Himpunan simbol terminal berhingga,

R = Himpunan aturan produksi,

S = Simbol start (non-terminal).

Salah satu algoritma *parsing* untuk CFG adalah algoritma CYK (Cocke, Younger, Kasami). Algoritma ini memiliki *pre-requisite*, yakni CFG haruslah diubah menjadi bentuk *Chomsky Normal Form* (CNF). Pada Teori Bahasa Formal, suatu CFG dikatakan dalam bentuk *Chomsky Normal Form* apabila seluruh aturan produksinya berbentuk setidaknya salah satu dari berikut.

$$A \rightarrow BC,$$

$$A \rightarrow \alpha,$$

$$S \rightarrow \epsilon$$

A = Simbol non-terminal,

B = Simbol non-terminal,

C = Simbol non-terminal,

α = Simbol terminal,

S = Simbol start (non-terminal),

ϵ = String kosong.

Proses penyederhanaan CFG ini bertujuan untuk simplifikasi *grammar* dengan menghilangkan segala aturan produksi yang tidak berarti. Langkah penyederhanaan CFG menjadi CNF secara umum terdiri dari empat langkah. Misalkan suatu CFG sebagai berikut.

$$S \rightarrow ASB$$
$$A \rightarrow aAS \mid a \mid \varepsilon$$
$$B \rightarrow SbS \mid A \mid bb$$

Pertama, lakukan eliminasi *Start Symbol* (S) pada ruas kanan produksi. Bentuklah simbol baru, misalkan S0, yang berperan sebagai *Start Symbol* yang baru.

$$S0 \rightarrow S$$
$$S \rightarrow ASB$$
$$A \rightarrow aAS \mid a \mid \varepsilon$$
$$B \rightarrow SbS \mid A \mid bb$$

Lalu, hilangkan seluruh produksi yang menghasilkan *null,* maka *grammar* menjadi sebagai berikut.

$$S0 \rightarrow S$$
$$S \rightarrow AS \mid ASB \mid SB \mid S$$
$$A \rightarrow aAS \mid aS \mid a$$
$$B \rightarrow SbS \mid A \mid bb$$

Hilangkan seluruh *unit production,* yakni produksi non-terminal yang menghasilkan non-terminal. *Grammar* di atas menjadi sebagai berikut.

$$S0 \rightarrow AS \mid ASB \mid SB$$
$$S \rightarrow AS \mid ASB \mid SB$$
$$A \rightarrow aAS \mid aS \mid a$$
$$B \rightarrow SbS \mid bb \mid aAS \mid aS \mid a$$

Langkah selanjutnya, eliminasi seluruh terminal dari ruas kanan jika berdampingan dengan terminal lain atau non-terminal.

$$S0 \rightarrow AS \mid ASB \mid SB$$

$$S \rightarrow AS \mid ASB \mid SB$$

$$A \rightarrow XAS \mid XS \mid a$$

$$B \rightarrow SYS \mid VV \mid XAS \mid XS \mid a$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$V \rightarrow b$$

Langkah terakhir adalah eliminasi seluruh produksi yang ruas kanannya memiliki simbol non-terminal lebih dari dua.

$$S0 \rightarrow AS \mid PB \mid SB$$

$$S \rightarrow AS \mid QB \mid SB$$

$$A \rightarrow RS \mid XS \mid a$$

$$B \rightarrow TS \mid VV \mid US \mid XS \mid a$$

$$X \rightarrow a$$

$$Y \rightarrow b$$

$$V \rightarrow b$$

$$P \rightarrow AS$$

$$Q \rightarrow AS$$

$$R \rightarrow XA$$

$$T \rightarrow SY$$

$$U \rightarrow XA$$

CFG sudah dalam bentuk CNF dan siap untuk di-*parse* dengan algoritma CYK.[1]

[1] https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/

3. Syntax Javascript

Syntax Javascript mendefinisikan *nilai* dari suatu data menjadi dua bagian, yakni *Fixed* dan *Variable values*. *Fixed values* atau *Literals* merupakan data statik berupa nilai baku, contohnya seperti *integer* '10' atau *string* "Hello world!", sedangkan *Variable values* merupakan tempat penyimpanan nilai dari suatu data, diimplementasikan dengan deklarasi dan *assignment* terhadap suatu nilai. Deklarasi variabel menggunakan satu dari tiga *keywords,* yakni *var, let,* atau *const.*

Pada Bahasa Pemrograman Javascript, terdapat lima jenis operator yang terdefinisi, yakni operator *assignment, arithmetic, comparison, bitwise, logical,* serta *type operator. Assignment operators* berperan dalam penempatan nilai data ke dalam suatu variabel, *comparison operators* dalam perbandingan antara dua nilai*, bitwise operators* digunakan untuk operasi bit-level, *logical operators* berupa operator perbandingan boolean, serta *type operators* untuk penentuan tipe data[2].

Javascript merupakan bahasa yang *case-sensitive* sehingga penggunaan huruf kapital harus diperhatikan dalam penulisan *identifier*. *Identifier* merupakan penamaan variabel, *keyword*, atau fungsi yang memiliki aturan penulisan tertentu layaknya bahasa pemrograman pada umumnya, yakni haruslah diawali oleh salah satu dari karakter berikut.

- Karakter alphabet (A-Z atau a-z),
- Dollar *sign* ($),
- Underscore (_)

Bahasa Javascript menerima penulisan yang diambil dari himpunan karakter *unicode*, mencakup hampir segala bentuk karakter, tanda baca, serta simbol-simbol yang umum digunakan.

[2] https://www.w3schools.com/js/js_operators.asp

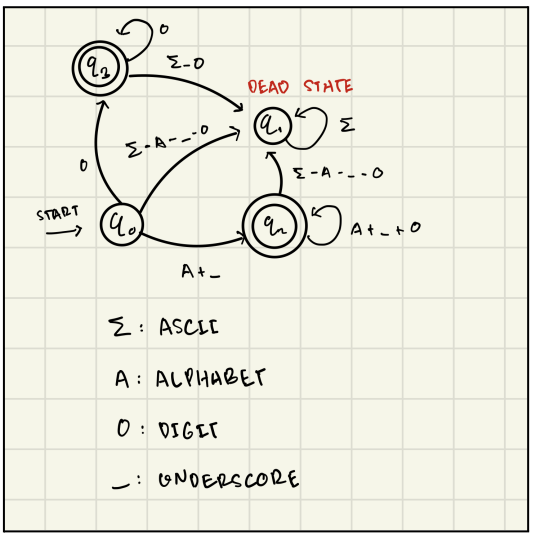
BAB II
HASIL FA DAN CFG

1. FA

Finite automata pada program kami digunakan untuk melakukan validasi terhadap nama variabel. *Finite automata* menerima semua *string* yang memenuhi aturan penamaan variabel pada bahasa pemrograman Node JS. Alfabet dari *finite automata* kami mencakup semua karakter ASCII. Terdapat empat *state* yang terdapat di dalam *finite automata* dengan rincian sebagai berikut:

- q0 : *State* awal sebelum membaca *string* (*Start state*)
- q1 : *State* mati ketika terdapat pembacaan karakter yang tidak sesuai aturan
- q2 : *State* ketika menerima pembacaan nama variabel (*Final state*)
- q3 : *State* ketika menerima pembacaan sebuah *number* (*Final state*)

Transisi yang terdapat di dalam *finite automata* dapat dimodelkan dengan diagram transisi sebagai berikut:



Finite automata tersebut diimplementasikan dengan fungsi finite_automata yang akan menghasilkan state akhir dari sebuah pembacaan *string*. Penentuan apakah sebuah *string* merupakan nama variabel yang valid atau tidak ditentukan pada fungsi convert_input dengan memeriksa apakah state yang dihasilkan oleh *finite automata* merupakan *final state* atau bukan.

2. CFG

Context-free Grammar untuk bahasa pemrograman Javascript telah disimpan pada file CFG.txt di dalam folder config. Berikut adalah hasil penjabarannya.

```
S → SINGULAR_STATEMENT S | EXPRESSION ; S | @epsilon
SINGULAR_STATEMENT → FUNCTION_DEC | DECLARATION ; |
IF_STATEMENT | WHILE_LOOP | DO_WHILE_LOOP | FOR_LOOP |
SWITCH_STATEMENT | TRY_STATEMENT | THROW_STATEMENT ; |
RETURN_STATEMENT ; | BREAK ; | CONTINUE ; | DELETE ; |
@epsilon
IF_STATEMENT → if ( EXPRESSION ) STATEMENT | if ( EXPRESSION
) STATEMENT else STATEMENT
WHILE_LOOP → while ( EXPRESSION ) STATEMENT
DO_WHILE_LOOP → do STATEMENT while ( EXPRESSION )
FOR_LOOP → for ( FOR_INIT ; EXPRESSION ; EXPRESSION )
STATEMENT | for ( FOR_INIT ; EXPRESSION ; ASSIGNMENT_EXP )
STATEMENT
FOR_INIT → ASSIGNMENT_EXP | DECLARATION | EXPRESSION
SWITCH_STATEMENT → switch ( EXPRESSION ) { CASE_STATEMENT }
CASE_STATEMENT → case EXPRESSION : S BREAK CASE_STATEMENT |
case EXPRESSION : S CASE_STATEMENT | DEFAULT_CASE | @epsilon
DEFAULT_CASE → default : S BREAK | default : S
TRY_STATEMENT → try { S } catch { S } | try { S } catch (
@varname ) { S } | try { S } finally { S } | try { S } catch
{ S } finally { S } | try { S } catch ( @varname ) { S }
finally { S }
THROW_STATEMENT → throw EXPRESSION
RETURN_STATEMENT → return | return RETURN_EXPRESSION
RETURN_EXPRESSION → EXPRESSION | EXPRESSION ,
RETURN_EXPRESSION
BREAK → break
CONTINUE → continue
DELETE → delete EXPRESSION
STATEMENT → { S } | SINGULAR_STATEMENT
DECLARATION → ASSIGNMENT_EXP | LET_DEC | VAR_DEC | CONST_DEC
| FUNCTION_DEC
LET_DEC → let @varname | let LEFT_HAND_EXP = EXPRESSION |
let @varname , DEC_ASSIGNMENT | let LEFT_HAND_EXP =
EXPRESSION , DEC_ASSIGNMENT
VAR_DEC → var @varname | var LEFT_HAND_EXP = EXPRESSION |
var @varname , DEC_ASSIGNMENT | var LEFT_HAND_EXP =
EXPRESSION , DEC_ASSIGNMENT
CONST_DEC → const @varname = EXPRESSION | const {
DESCTRUCTURE } = EXPRESSION | const @varname = EXPRESSION ,
CONSTDEC_ASSIGNMENT | const { DESCTRUCTURE } = EXPRESSION ,
CONSTDEC_ASSIGNMENT
DEC_ASSIGNMENT → @varname | LEFT_HAND_EXP = EXPRESSION |
```

```
@varname , DEC_ASSIGNMENT | LEFT_HAND_EXP = EXPRESSION ,
DEC_ASSIGNMENT
DESCTRUCTURE → @varname | @varname , DESCTRUCTURE
ASSIGNMENT_EXP  → LEFT_HAND_EXP = EXPRESSION
CONSTDEC_ASSIGNMENT → @varname = EXPRESSION | @varname =
EXPRESSION , CONSTDEC_ASSIGNMENT
FUNCTION_DEC → function @varname ( FUNC_PARAMS ) { S } |
function @varname ( ) { S }
FUNC_PARAMS → @varname | { DESCTRUCTURE } | FUNC_PARAMS ,
FUNC_PARAMS | @epsilon
BOOLEAN → true | false
LEFT_HAND_EXP → { DESCTRUCTURE } | @varname | EXPRESSION .
LEFT_HAND_EXP
EXPRESSION → EXPRESSION . EXPRESSION | @varname ( ) |
@varname ( FUNC_CALL_PARAMS ) | { OBJ_DEC } | BOOLEAN | null
| undefined | @value | @varname | ( EXPRESSION ) | EXPRESSION
+ EXPRESSION | EXPRESSION @unary_op | @unary_op EXPRESSION |
EXPRESSION ? EXPRESSION : EXPRESSION | ASSIGNMENT_EXP
FUNC_CALL_PARAMS →  EXPRESSION | FUNC_CALL_PARAMS ,
FUNC_CALL_PARAMS | @epsilon
OBJ_DEC → @varname : EXPRESSION | @varname : EXPRESSION ,
OBJ_DEC | @epsilon
```

BAB III
IMPLEMENTASI DAN PENGUJIAN

1. Spesifikasi Teknis Program

Struktur data yang digunakan dalam parsing Bahasa Pemrograman Javascript antara lain *map* untuk memetakan produksi dari grammar CFG dan CNF, array untuk menyimpan *stream* dari input file, serta *matrix of sets* untuk CYK *parsing algorithm.* Adapun struktur folder yang digunakan adalah sebagai berikut.

```
tubes-tbfo-bob
├── config
│   └── CFG.txt
├── out.txt
├── src
│   ├── CYK.py
│   ├── cfg_to_cnf.py
│   ├── finite_automata.py
│   ├── input_converter.py
│   ├── main.py
│   ├── testing.py
│   └── util.py
└── test
    ├── test_const_acc.js
    ├── test_const_rej.js
    ├── test_for_acc.js
    ├── test_for_rej.js
    ├── test_func_acc.js
    ├── test_func_rej.js
    ├── test_if_acc.js
    ├── test_if_rej.js
    ├── test_switch_acc.js
    ├── test_switch_rej.js
    ├── test_try_acc.js
    ├── test_try_rej.js
    ├── test_while_acc.js
    └── test_while_rej.js
```

Algoritma yang digunakan pada *source code* adalah sebagai berikut.

1. **cfg_to_cnf.py**
   File ini berisi algoritma untuk mengubah *Context-free Grammar* menjadi *Chomsky Normal Form*.

```
function cfgtext_to_dict(filepath : string) → map
{ Menghasilkan map berupa pasangan production dari CFG
pada file masukan }
```

```
function remove_nullables(rules : array of array of
string, nullables : array of string) → array of array of
string
{ Menghilangkan epsilon production dari rules pada CFG }
```

```
function isTerminal(str : string) → boolean
{ Mengembalikan True apabila string merupakan terminal }
```

```
function cfg_to_cnf(filepath : string) → map
{ Menghasilkan map berupa pasangan production dari CNF
yang dihasilkan oleh CFG dari file masukan }
```

2. **CYK.py**
   File ini berisi algoritma CYK untuk *parsing* kode Javascript yang sudah dipecah menjadi array *stream* dari file masukan.

```
function cyk_parse (word : array, cnf : map) → boolean
{ Melakukan algoritma CYK untuk parsing input word
berdasarkan input grammar Chomsky Normal Form,
mengembalikan True untuk Accept dan False untuk reject. }
```

3. **finite_automata.py**
   File ini berisi fungsi untuk mensimulasikan sebuah *finite automata* yang akan menerima sebuah string dan menghasilkan *final state* dari *finite automata* tersebut. Fungsi transisi didefinisikan pada variabel _TRANSISITIONS menggunakan *dictionary*

```
function finite_automata (input_str : string) → string
{ Menghasilkan state pada finite automata setelah menerima
input_str }
```

4. **input_converter.py**
   File ini mengubah string dalam bahasa pemrograman Node JS dan menghasilkan sebuah array yang berisi token dari string awal. Fungsi ini menghilangkan *comment* yang ada pada Node JS dan mengubah beberapa string pada input agar dapat dibaca pada algoritma CYK.

```
function convert_input(input_string : string) → array
{ Menghasilkan array yang berisi token dari input_string }
```

5. **util.py**
   File ini berisi animasi-animasi sebagai pemanis dalam main program.

```
procedure loading(input message : string)
{ I.S String message terdefinisi
  F.S Ditampilkan loading animation dengan message sebagai
  pesan loading }
```

```
procedure analyzing()
{ I.S State program sembarang
  F.S Ditampilkan animasi analyzing }
```

```
procedure splash()
{ I.S State program sembarang
  F.S Ditampilkan splash screen }
```

6. **main.py**
   File ini berisi program utama untuk parsing kode Javascript.

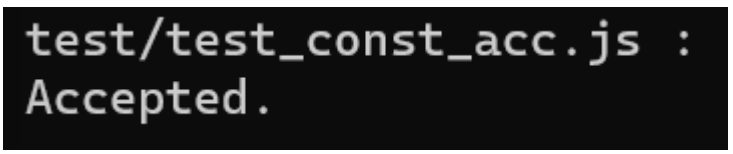
```
procedure main()
{ I.S Program parser belum dimulai
  F.S Program selesai dijalankan }
```

2. Uji Kasus

Test Case #1 (test_const_acc.js)

```
const x = 2;
document.getElementById("demo").innerHTML = "Hello Dolly.";
let x;
var testing = [1, 2, 3];
```

```
Verdict: Accepted
```



Test Case #2 (test_const_rej.js)

```
document.getElementById("demo")innerHTML = "Hello Dolly.";
```

Verdict: Syntax error
(Syntax error karena seharusnya terdapat ‘.’ sebelum ‘innerHTML’)

```
test/test_const_rej.js :
Syntax error.
```

Test Case #3 (test_for_acc.js)

```
i = 3;
for(i = 1; i ⩽ 5; i+=1) {
  console.log(i);
}
```

Verdict: Accept

```
test/test_for_acc.js :
Accepted.
```

Test Case #4

```
for(let i = 2 i ⩽ 100; i++) {
  console.log(i);
}

i = 3;
for(i; i ⩽ 5; i+=1) {
  console.log(i);
}
```

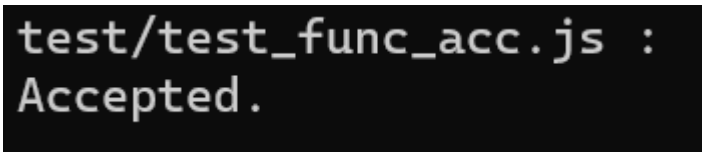
Verdict: Syntax error
(Syntax error karena pada baris pertama setelah ‘i = 2’ seharusnya terdapat ‘;’)

```
test/test_for_rej.js :
Syntax error.
```

Test Case #5

```
function testing(x) {
  switch(x) {
  case 0:
    return 4;
  case 2:
    return "hello";
  default:
    return [1, 2, 3];
  }
}
```

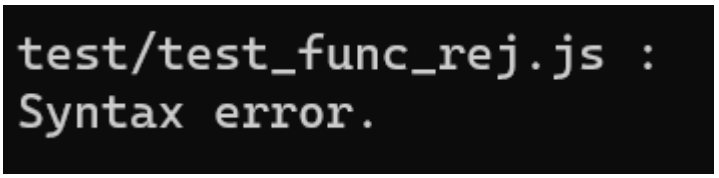
Verdict: Accepted



Test Case #6

```
function testing(x) {
  switch(x) {
  case 0:
    ret 4;
  case 2:
    return "hello";
  default:
    return [1, 2, 3];
  }
}
```

Verdict: Syntax error
(Syntax error karena pada ‘case 0’ terdapat ‘ret’)



Test Case #7

```
if(x == 4) {
```

```
  console.log("four");
} else if(x ⩽ 7) {
  console.log("seven");
} if(true) {
  console.log("true");
} else if(null) {
  console.log("null");
} else {
  console.log("congrats");
}
```

Verdict: Accepted

```
test/test_if_acc.js :
Accepted.
```

Test Case #8

```
if(x == 4) {
  console.log("four");
} else if(x ⩽ 7) {
  console.log("seven");
} else (true) {
  console.log("true");
} else if(null) {
  console.log("null");
} else {
  console.log("congrats");
}
```

Verdict: Syntax error
(Syntax error karena setelah ‘else’ pada baris 5 terdapat ‘else if’ yang tidak didahului oleh ‘if’)

```
test/test_if_rej.js :
Syntax error.
```

Test Case #9

```
switch(x) {
  case 0:
    y = 4;
    break;
  case 2:
    y = 5;
    break;
  default:
    y = 10;
}
```

Verdict: Accepted

```
test/test_switch_acc.js :
Accepted.
```

Test Case #10

```
switch(x)
  case 0:
    y = 4;
    break;
  case2:
    y = 5;
    break;
  default:
    y = 10;
}
```

Verdict: Syntax error
(Syntax error karena ‘case2’ seharusnya ‘case 2’)

```
test/test_switch_rej.js : Syntax error.
```

Test Case #11

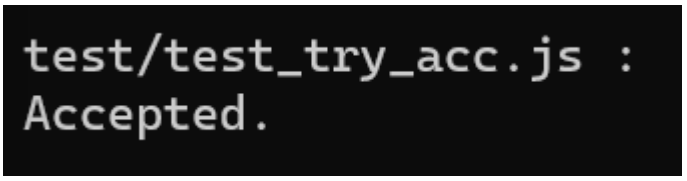
```
try {
  let x = 5;
  while(x =≠ 0) {
```

```
    console.log(x);
    x--;
  }

  if(false) {
    throw "Hello";
  }

  delete x;
} catch(err) {
  console.log(err);
} finally {
  y += 3;
}
```

Verdict: Accepted



Test Case #12

```
try {
  let x = 5;
  while(x =≠ 0) {
    console.log(x);
    x--;
  }

  if(false) {
    throw "Hello";
  }

  delete x;
}
```

Verdict: Syntax error
(Syntax error karena ‘try’ seharusnya diikuti dengan ‘catch’ atau ‘finally’)

```
test/test_try_rej.js :
Syntax error.
```

Test Case #13

```
let x = 6;
while(x ⩽ 100) {
  if (x ⩵ 5) {
    break;
  }
  x++;
}
```

Verdict: Accepted

```
test/test_while_acc.js :
Accepted.
```

Test Case #14

```
let x = 6;
while(x ⩽ 100)
  if (x ⩵ 5) {
    break;
  }
  x++;
}
```

Verdict: Syntax error
(Syntax error karena tidak terdapat ‘{‘ yang seharusnya ada pada akhir baris 2)

```
test/test_while_rej.js :
Syntax error.
```

BAB IV
PEMBAGIAN TUGAS

No.	Nama	Tugas
1	Rinaldy Adin	Convert CFG to CNF, Grammar
2	Enrique Alifio Ditya	CYK parsing, Main program, Laporan
3	Rava Maulana Azzikri	Finite automata, Input converter, Testing

LAMPIRAN

Repositori Github :
https://github.com/Rinaldy-Adin/tubes-tbfo-bob