LAPORAN TUGAS BESAR

TEORI BAHASA FORMAL DAN AUTOMATA

PARSER BAHASA JAVASCRIPT

Diajukan untuk memenuhi tugas besar mata kuliah IF2124 Teori Bahasa Formal dan Automata



Disusun oleh:

Bintang Hijriawan Jachja (13521003)

Muhammad Zulfiansyah Bayu Pratama (13521028)

Fahrian Afdholi (13521031)

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

DAFTAR ISI

[**BAB I** 3](#_Toc120298389)

[**DESKRIPSI MASALAH** 3](#_Toc120298390)

[BAB II 4](#_Toc120298391)

[TEORI DASAR 4](#_Toc120298392)

[2.1 JAVASCRIPT 4](#_Toc120298393)

[2.2 CONTEXT-FREE GRAMMAR 4](#_Toc120298394)

[2.3 CHOMSKY NORMAL FORM 5](#_Toc120298395)

[2.4 COCKE-YOUNGER-KASAMI 6](#_Toc120298396)

[BAB III 8](#_Toc120298397)

[CFG 8](#_Toc120298398)

[3.1 CFG 8](#_Toc120298399)

[3.1.1 Variables (V) 8](#_Toc120298400)

[3.1.2 Terminals (T) 8](#_Toc120298401)

[3.1.3 Production (P) 8](#_Toc120298402)

[3.1.4 Start Symbol (S) 9](#_Toc120298403)

[BAB IV 10](#_Toc120298404)

[IMPLEMENTASI DAN PENGUJIAN 10](#_Toc120298405)

[4.1 Implementasi 10](#_Toc120298406)

[4.1.1 CFGtoCNF.py 10](#_Toc120298407)

[4.1.2 parser.py 10](#_Toc120298408)

[4.1.3 token.py 10](#_Toc120298409)

[4.1.4 main.pys 11](#_Toc120298410)

[4.2 Eksperimen 11](#_Toc120298411)

[4.2.1 inputAcc.js 11](#_Toc120298412)

[4.2.2 inputReject.js 12](#_Toc120298413)

[4.2.3 sample\_07.js 13](#_Toc120298414)

[BAB V 14](#_Toc120298415)

[LAMPIRAN 14](#_Toc120298416)

[5.1 Repository Github 14](#_Toc120298417)

[5.2 Pembagian Tugas 14](#_Toc120298418)

# **BAB I**

## **DESKRIPSI MASALAH**

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks bahasa atau parsing yang dibuat oleh programmer untuk memastikan program dapat dieksekusi tanpa menghasilkan error. Parsing ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa berjenis interpreter maupun compiler, keduanya pasti melakukan pemeriksaan sintaks. Perbedaannya terletak pada apa yang dilakukan setelah proses pemeriksaan (kompilasi/compile) tersebut selesai dilakukan.

Dibutuhkan grammar bahasa dan algoritma parser untuk melakukan parsing. Sudah sangat banyak grammar dan algoritma yang dikembangkan untuk menghasilkan compiler dengan performa yang tinggi. Terdapat CFG, CNF-e, CNF+e, 2NF, 2LF, dll untuk grammar yang dapat digunakan, dan terdapat LL(0), LL(1), CYK, Earley’s Algorithm, LALR, GLR, Shift-reduce, SLR, LR(1), dll untuk algoritma yang dapat digunakan untuk melakukan parsing.

Pada tugas besar ini, kami diminta untuk mengimplementasikan parser untuk JavaScript (Node.js) untuk beberapa statement dan sintaks bawaan JavaScript dengan menggunakan konsep CFG untuk pengerjaan parser yang mengevaluasi syntax program.

# BAB II

## TEORI DASAR

### 2.1 JAVASCRIPT

JavaScript adalah bahasa pemrograman tingkat tinggi yang pada awalnya dikembangkan untuk membuat website menjadi lebih “hidup”. Bersama dengan HTML dan CSS, JavaScript menjadi bahasa pemrograman paling populer untuk mengembangkan aplikasi berbasis web. Bahasa ini mampu memberikan logic ke dalam website, sehingga website tersebut memiliki fungsionalitas tambahan dan lebih interaktif. Namun, developer dapat menggunakan bahasa pemrograman JavaScript di berbagai lingkungan pengembangan. Sehingga, tidak hanya sebatas browser/client, namun JavaScript juga bisa berjalan di server menggunakan Node.js. JavaScript merupakan bahasa pemrograman jenis interpreter berorientasi objek.

Dalam proses pembuatan program menjadi *executable*, terjadi pemeriksaan *syntax* untuk melakukan pemeriksaan apakah bahasa dalam kode tersebut termasuk kedalam bahasa JavaScript. Di bawah ini merupakan contoh-contoh kata kunci bawaan bahasa pemrograman JavaScript yang memiliki format penulisan yang harus diperiksa.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| break | const | case | catch | continue |
| default | delete | else | false | finally |
| for | function | if | let | null |
| return | switch | throw | try | true |
| var | while |  |  |  |

Tabel 2.1.1 Kata Kunci Bawaan Bahasa Pemrograman JavaScript

### 2.2 CONTEXT-FREE GRAMMAR

CFG atau Context Free Grammar adalah tata bahasa formal di mana setiap aturan produksi adalah dalam bentuk A → B di mana A adalah pemproduksi, dan B adalah hasil produksi. Batasannya hanyalah ruas kiri adalah sebuah simbol variabel. Dan pada ruas kanan bisa berupa terminal, symbol, variable ataupun ɛ, Contoh aturan produksi yang termasuk CFG adalah seperti berikut ini:

X → bY | Za

Y → aY | b

Z → bZ | ɛ

CFG adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa.

Ada 4 komponen yang dapat merepresentasikan suatu bahasa, diantaranya:

1. Himpunan berhingga dari simbol-simbol yang membentuk string bahasa. Simbol-simbol ini biasa disebut dengan terminal.
2. Himpunan berhingga dari variabel atau non-terminal, dimana setiap variable merepresentasikan himpunan string yang menjadi bagian dari bahasa yang didefinisikan.
3. Himpunan berhingga produksi atau aturan.
4. Start symbol.

Kita bisa merespresentasikan sebuah CFG G dengan empat komponen diatas dengan notasi

G = {V, T, P, S}

G = Context-Free Grammar

V = himpunan berhingga simbol-simbol

T = himpunan berhingga variabel

P = himpunan berhingga produksi

S = start symbol

### 2.3 CHOMSKY NORMAL FORM

CNF atau Chomsky Normal Form adalah salah satu bentuk CFG dimana CFG G tidak memiliki ɛ-*production*, *unit production* atau *useless symbols.* Semua produksi dalam CNF akan berbentuk salah satu diantara dua bentuk berikut:

1. A -> B | C. A, B, dan C adalah variabel, atau
2. A -> a. A adalah variabel dan a adalah terminal.

Kita dapat mengubah CFG menjadi CNF dengan memenuhi kedua syarat diatas, yaitu semua produksi dengan panjang lebih dari sama dengan 2 diubah menjadi variabel dan semua produksi dengan panjang lebih dari sama dengan 3 dipecah menjadi beberapa produksi dengan panjang maksimal 2. Misalkan ada sebuah grammar dengan aturan produksi:

1. S -> AB | B | b
2. A -> a
3. B -> BA | BB | b

Agar *grammar* tersebut memenuhi syarat CNF yang pertama, kita harus mengubah semua produksi dengan panjang lebih dari sama dengan 2 menjadi variabel. Hasilnya sebagai berikut:

1. S -> AB | B | C
2. A -> a
3. B -> BA | BB | C

Lalu, kita harus memecah produksi dengan panjang lebih dari sama dengan 3 menjadi beberapa produksi yang lebih kecil, sehingga *grammar* yang baru adalah sebagai berikut:

1. S -> AB | D
2. A -> a
3. B -> BA | E
4. C -> b
5. D -> B | C
6. E -> BB | C

*Grammar* diatas adalah salah satu contoh *grammar* dalam bentuk CNF.

### 2.4 COCKE-YOUNGER-KASAMI

Cocke-Younger-Kasami atau CYK adalah sebuah algoritma parsing. Untuk dapat menggunakan algoritma ini, *grammar* yang digunakan harus dalam bentuk CNF. Algoritma menggunakan *dynamic programming* untuk menentukan apakah suatu *string* termasuk kedalam suatu bahasa atau tidak. Algoritma CYK sebagai berikut:

1. Diberikan suatu aturan produsksi suatu bahasa dalam bentuk CNF sebagai berikut:

S -> AB | BC

A -> BA | a

B -> CC | b

C -> AB | a

1. Akan dilakukan pemeriksaan apakah string “baaba” termasuk kedalam bahasa tersebut dengan menggunakan algoritma CYK.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | b | a | a | b | a |
| b | {B} | {S, A} | - | - | {S, A, C} |
| a |  | {A, C} | {B} | {B} | {S, A, C} |
| a |  |  | {A, C} | {S, C} | {B} |
| b |  |  |  | {B} | {S, A} |
| a |  |  |  |  | {A, C} |

Dapat dilihat bahwa S ada di kotak baris 1 kolom n, dengan n adalah panjang string. Sehingga “baaba” merupakan bagian dari bahasa tersebut.

# BAB III

## CFG

### 3.1 CFG

Berikut ini adalah CFG yang kami buat.

G = (V, T, P, S)

#### 3.1.1 Variables (V)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S | DEFAULT | CASE\_STATEMENT | SUBSWITCH | ARITHMETIC\_OPERATOR | MODEQ | VARIABLE | PARAM |
| SS | TRY | DEFAULT\_STATEMENT | SUBTHROW | B\_VAR | POWEQ | VALUE | IS\_EQUAL |
| BREAK | CATCH | TRY\_STATEMENT | ASSIGNMENT | DECLARATION | INCRE | SEMICOLON | IS\_EQUALS |
| CONTINUE | FINALLY | CATCH\_STATEMENT | CONDITION | RETURN | DECRE | DOUBLE\_DOT | IS\_NOT\_EQUAL |
| FOR | THROW | FINALLY\_STATEMENT | ADD | DELETE | ASSIGN\_OPERATOR | DOT | IS\_NOT\_EQUALS |
| WHILE | FUNC | FUNCTION\_ STATEMENT | SUBTRACT | EQUAL | VARBS | LCR | G |
| IF | WHILE\_STATEMENT | IF\_ELSE | MULT | PLUSEQ | NULL | RCR | L |
| ELSE | FOR\_STATEMENT | SUBREAK | DIV | SUBEQ | LET | LPR | GE |
| SWITCH | IF\_STATEMENT | SUBCONTINUE | MOD | MULTEQ | VAR | RPR | LE |
| CASE | SWITCH\_STATEMENT | SUBPROGRAM | POWER | DIVEQ | CONST | RELATIONAL\_OPERATOR |  |

#### 3.1.2 Terminals (T)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| break | case | add | equal | null | false | rpr | ge |
| continue | default | subtr | pluseq | let | semicolon | is\_equal | le |
| for | try | mult | subuseq | var | double\_dot | is\_equals |  |
| while | catch | div | multeq | const | dot | is\_not\_equal |  |
| if | finally | mod | diveq | variable | lcr | is\_not\_equals |  |
| else | throw | return | modeq | value | rcr | g |  |
| switch | func | delete | poweq | true | lpr | l |  |

#### 3.1.3 Production (P)

Produksi yang kami buat sebanyak 154 baris tertera pada file *bin/cfg.txt*

#### 3.1.4 Start Symbol (S)

Start symbol yang kami pakai adalah ‘S’ sehingga pada pengujian algoritma CYK, jika ‘S’ ada di tabel indeks (0, n-1) berarti string diterima.

# BAB IV

## IMPLEMENTASI DAN PENGUJIAN

### 4.1 Implementasi

#### 4.1.1 CFGtoCNF.py

File CFGtoCNF.py berisi fungsi-fungsi dan prosedur yang dibutuhkan untuk membaca sebuah file berisi CFG dan mengubah CFG menjadi CNF sehingga bisa digunakan untuk algoritma CYK.

|  |  |  |
| --- | --- | --- |
| NO | FUNGSI/PROSEDUR | KETERANGAN |
| 1 | readFile | Membaca file CFG dan memasukannya kedalam array sesuai format penulisan yang kami sepakati |
| 2 | addRule | Menambahkan suatu rule kedalam dictionary semetara untuk mencatat rule-rule yang sudah dibaca |
| 3 | convertCFG | Mengubah CFG yang berbentuk array menjadi CNF dalam bentuk array |
| 4 | mapCNF | Mengubah array CNF menjadi dictionary |
| 5 | writeCNF | Menuliskan array CNF kedalam sebuah file teks |

#### 4.1.2 parser.py

File parser.py berisi prosedur yang menjalankan algoritma CYK, lalu melakukan pemeriksaan apakah string yang dideteksi termasuk kedalam CFG yang sudah didefinisikan.

|  |  |  |
| --- | --- | --- |
| NO | FUNGSI/PROSEDUR | KETERANGAN |
| 1 | parserr | Menerima parameter berupa token dari file yang akan diuji dan array CNF yang sudah didefinisikan |

#### 4.1.3 token.py

File token.py berisi fungsi dan prosedur yang berfungsi mengubah file yang akan diuji menjadi token-token yang telah didefinisikan sebelumnya.

|  |  |  |
| --- | --- | --- |
| NO | FUNGSI/PROSEDUR | KETERANGAN |
| 1 | strToTokens | Menerima parameter file yang sudah diubah menjadi string dan ekspresi token yang sudah didefinisikan, dan mengembalikan array token hasil terjemahan string file |
| 2 | createToken | Menerima parameter berupa path file yang akan diuji, membuka file dan memanggil fungsi lex untuk menerjemahkan file menjadi token. |

#### 4.1.4 main.pys

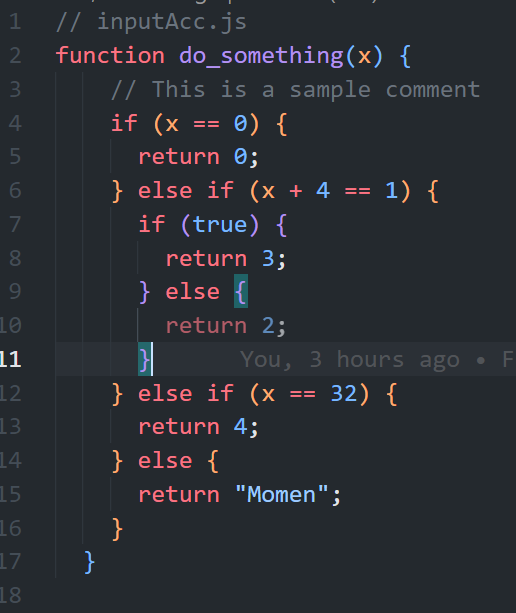
File main.py berisi driver utama yang menerima argumen path file main.py dan path file JavaScript yang akan diuji. Program lalu menampilkan splash screen yang sudah dibuat sebelumnya, dan melakukan seluruh prosedur algoritma CYK hingga mengeluarkan input “Accepted” atau “Syntax error” berdasarkan hasil uji. Dalam file ini, tidak didefinisikan fungsi ataupun prosedur.

### 4.2 Eksperimen

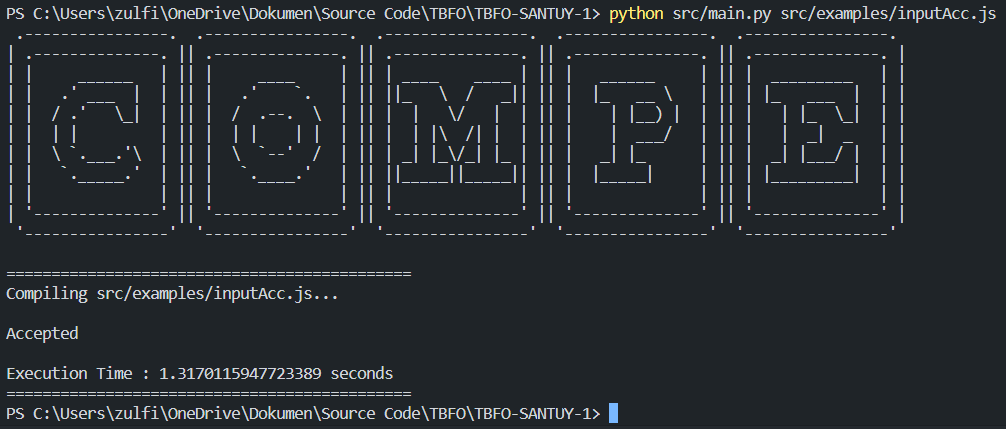
Dalam bagian ini, kami telah melakukan berbagai kasus uji berupa file javascript terhadap compiler bahasa Python yang telah kami buat. Kasus uji yang kami gunakan adalah sebagai berikut.

#### 4.2.1 inputAcc.js

Code :

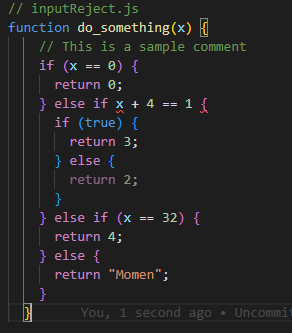


Hasil :

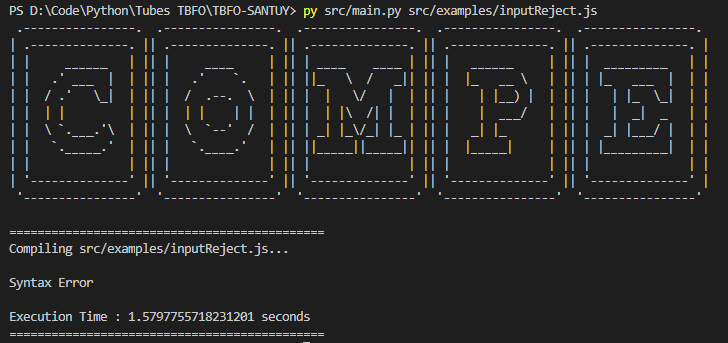


#### 4.2.2 inputReject.js

Code :

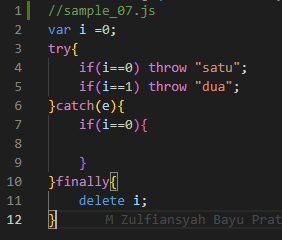


Hasil :

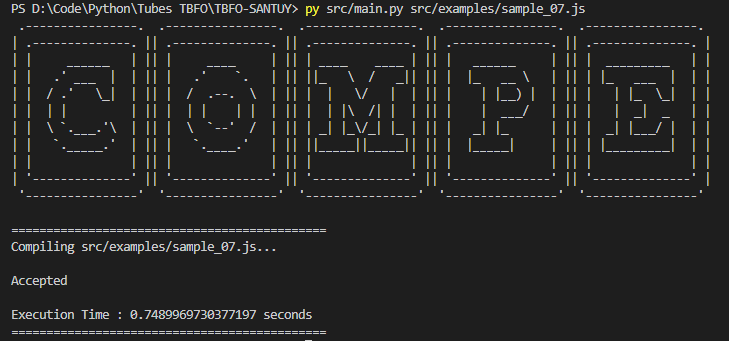


#### 4.2.3 sample\_07.js

Code:



Hasil



# BAB V

## LAMPIRAN

### 5.1 Repository Github

<https://github.com/zulfiansyah404/TBFO-SANTUY>

### 5.2 Pembagian Tugas

1. Bintang Hijriawan Jachja (13521003)

* CFGtoCNF.py
* Parser\_procces.py
* Main.py
* Laporan

1. Muhammad Zulfiansyah Bayu Pratama (1352028)

* Token.py
* Main.py
* Laporan

1. Fahrian Afdholi (13521031)

* Token.py
* CFG
* Laporan