PARSER NODE JS MENGGUNAKAN PYTHON

LAPORAN TUGAS BESAR

Disusun untuk memenuhi salah satu tugas besar
mata kuliah Teori Bahasa Formal dan Otomata
IF2124-03
Oleh

Hidayatullah Widan Ghaly B.	**13521015**
Bernardus Willson	**13521021**
Raditya Naufal A.	**13521022**

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Node JS merupakan sebuah lingkungan server *open source* yang dibuat pada tahun 2009 oleh Ryan Dahl. Pada awalnya node js hanya bisa digunakan dengan linux dan mac OS X yang kemudian terus dikembangkan hingga bisa berjalan di berbagai platform secara bersamaan. Node JS mendukung pembuatan web server dan networking tools dengan menggunakan JavaScript dan berbagai modul untuk mengatur fungsionalitas utama.

Dalam proses pembuatan web/program, ada sebuah proses yang mengubah suatu bahasa menjadi sebuah instruksi yang dapat dieksekusi mesin. Saat proses itu terjadilah pengecekan sintaks yang dibuat oleh pengembang bahasa tersebut. Pengecekan sintaks ini bertujuan untuk memastikan bahwa perintah yang dibuat sesuai dengan kaidah bahasa yang digunakan. Setiap bahasa pemrograman yang dibuat pasti akan memiliki tahap pengecekan sintaksnya masing-masing.

Sebuah grammar dan algoritma parsing dibutuhkan untuk melakukan pengecekan sintaks. Pada tugas besar ini, kami mengimplementasikan compiler untuk node js yang akan melakukan pemeriksaan terhadap program yang dibuat oleh user terhadap sintaks yang digunakan oleh node js. Pemeriksaan sintaks ini dilakukan dengan konsep CFG yang telah dipelajari sebelumnya pada mata kuliah Teori Bahasa Formal dan Otomata.

BAB II
DASAR TEORI

2.1. Context Free Grammar

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

$$X \rightarrow bY \mid Za$$

$$Y \rightarrow aY \mid b$$

$$Z \rightarrow bZ \mid \varepsilon$$

CFG merupakan sebuah tata bahasa yang memiliki tujuan sama seperti tata bahasa biasa yaitu menjadi cara untuk menghasilkan suatu kalimat dalam suatu bahasa. Definisi formal dari CFG dapat dijabarkan sebagai berikut :

G = (V, T, P, S)

V = himpunan terbatas variabel

T = himpunan terbatas terminal

P = himpunan terbatas dari produksi

S = start symbol

CFG disini menjadi dasar dalam proses analisis sintaks. Bagian-bagian dari sintaks akan didefinisikan dalam CFG dan kemudian dibentuk menjadi parse tree yang digunakan untuk menggambarkan variabel sebagai terminal. Setiap variabel akan diturunkan hingga tidak ada lagi variabel yang tersisa.

Proses parsing pada umumnya terbagi menjadi 2 tipe yaitu Leftmost Derivation dan Rightmost Derivation. Perbedaan kedua hal tersebut hanyalah dari mana arah penggantian variabel dengan terminalnya.

2.2. Chomsky Normal-Form

CNF atau Chomsky Normal-Form merupakan bentuk normal dari CFG. CNF dibuat dengan menggunakan CFG namun dilakukan penyederhanaan sehingga tahap poduksi yang "kurang penting", unit, dan ε dihilangkan. Hal tersebut perlu dilakukan karena CNF memiliki syarat yaitu tidak adanya *useless production,* tidak memiliki produksi unit, dan tidak memiliki ε. Bentuk dari produksi CNF sebagai berikut.

A → BC atau A → a

A = variabel

B = terminal

Dalam pembuatan CNF diperlukan langkah-langkah pembentukan yang dapat dideskripsikan sebagai berikut.

• Biarkan aturan produksi yang sudah dalam bentuk CNF.

• Penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan dengan panjang ruas kanannya lebih dari satu.

• Penggantian aturan produksi yang ruas kanannya memuat lebih dari dua simbol variable.

• Penggantian tersebut dilakukan berkali-kali sampai pada akhirnya aturan produksi memenuhi aturan dari CNF.

2.3. Cocke-Younger Kasami

Cocke younger kasami atau lebih dikenal dengan sebutan CYK merupakan sebuah algoritma parsing yang digunakan dalam CFG. Untuk menggunakan CYK, grammar harus diubah ke dalam bentuk CNF. Algoritma ini menggunakan dynamic programming untuk menentukan apakah suatu string dapat dikategorikan sebuah grammar. Sebagai contoh, diberikan aturan produksi sebagai berikut :

S -> AB | BC

A -> BA | a

B -> CC | b

C -> AB | a

Dilakukan pengecekan terhadap string "baaba" dengan algoritma CYK, kemudian hasil pengecekan akan dimasukkan ke tabel.

	b	a	a	b	a
b	{B}	{S,A}	Φ	Φ	{S,A,C}
a		{A,C}	{B}	{B}	{S,A,C}
a			{A,C}	{S,C}	{B}
b				{B}	{S,A}
a					{A,C}

Tabel 2.3.1 Tabel pengecekan CYK

Dari tabel di atas, dapat dilihat bahwa start symbol berada pada (1,n) dimana n merupakan panjang string yang di cek. Karena start symbol ada pada (1,n) maka string “baaba” memenuhi aturan produksi grammar.

2.4. Bahasa Pemrograman Node JS

Node JS merupakan sebuah lingkungan server *open source* yang dibuat pada tahun 2009 oleh Ryan Dahl. Pada awalnya node js hanya bisa digunakan dengan linux dan mac OS X yang kemudian terus dikembangkan hingga bisa berjalan di berbagai platform secara bersamaan. Node JS mendukung pembuatan web server dan networking tools dengan menggunakan JavaScript dan berbagai modul untuk mengatur fungsionalitas utama.

Salah satu fitur yang disediakan oleh node js adalah pemrograman asinkron. Karena hal tersebut node js menjadi lebih efektif dalam menjalankan sebuah server dibandingkan dengan PHP atau ASP. Node js menghilangkan waktu tunggu yang biasa ada di server dengan PHP atau ASP dan lanjut ke permintaan berikutnya. Node JS juga sangat efisien terhadap penggunaan memorinya.

Dalam prosesnya, terdapat pemeriksaan sintaks pada saat pengubahan bahasa pemrograman menjadi set instruksi yang dimengerti oleh komputer. Pengecekan sintaks ini bertujuan untuk memastikan bahwa instruksi sesuai dengan kaidah bahasa yang digunakan. Dalam hal ini node js sudah dibekali dengan pemeriksaan sintaks, jadi apabila ada kesalahan penulisan program tidak akan dijalankan dan diberi tahu dimana letak kesalahannya. DAlam node js terdapat beberapa kata kunci yang sudah memiliki fungsinya sendiri jadi tidak bisa

digunakan sebagai variabel dan kata kata tersebut akan digunakan sebagai terminal saat pemeriksaan sintaks. Kata kunci tersebut adalah.

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

Tabel 2.4.1 Tabel Kata Kunci Node JS

BAB III

ANALISIS PERSOALAN

3.1. CFG Production

Ini adalah hasil CFG yang telah kelompok kami buat untuk membuat sebuah compiler node js.

G = (V, T, P, S)

Variable/Non-Terminal Symbol(V)

EQUAL	ISEQ	ISEQQ	NEQQ	LB	RB	COLON
L	LE	POW	DIV	MUL	SUBTR	ADD
MOD	GE	G	NEQ	SUBTREQ	MULEQ	SUMEQ
ELSE	THEN	IF	NOT	OR	AND	DIVEQ
ELIF	WHILE	FALSE	BREAK	TRUE	AS	CLASS
IMPORT	FROM	FOR	FUNC	CONTINUE	ALET	AVAR
IN	RETURN	WITH	COMMA	DOTBETWEEN	DOT	LSB
TYPE	NEWLINE	VAR	INT	RCB	LCB	RSB
S	SS	PART	SENTENCE	VARINT	STATIC	AALET
PARAMDICT	DICT	MULTILINE	STRING	CURLY	CONST_STMT	AAVAR
LIST	ARIT_OP	ARIT_OPERATOR	LOGI_VAR	BOOL	LOGI_OP	AACONST
METHOD_STMT	STATIC	PARAMFUNGSI	VAR_FUNC	PARAM	WITH_STATE	LOGI_OPERATOR
SINGLE_SENTENCE	IF_STMT	ELIF_STMT	ELSE_STMT	ELIF_BANYAK	IF_BLOCK	ARRAY
FUNC_STMT	ASSIGNMENT	OPERATION	ASSIGN_OP	DECREMENT	INCREMENT	ASSIGN
ASSIGN_STMT	FUNC_SENTENCE	IF_STMT_FUNC	ELIF_STMT_FUNC	ELSE_STMT_FUNC	ELIF_BANYAK_FUNC	IF_BLOCK_FUNC
MULTI_CASE	DO	WHILE_STMT	CLASS_SENTENCE	CLASS_STMT	CLASS	FUNCTION
CASE_STMT	WHILE_STMT_FUNC	LOOP_SENTENCE_FUNC	FOR_STMT	SINGLE_LOOP_SENTENCE	LOOP_SENTENCE	IMPORT_STMT
DEFAULT_STMT	FINALLY	THROW	CATCH	TRY	ALL	ENTER

DEFAULT	SCOLON	SOLOVAR	SENTENCE_W_ENTER	TRY_STMT	SWITCH	CASE

Terminal Symbol(T)

iseq	equal	iseqq	neqq	lb	rb	colon
l	le	pow	div	mul	subtr	add
mod	ge	g	neq	subtreq	muleq	sumeq
else	then	if	not	or	and	diveq
while	"false	"true	do	break	as	class
return	for	func	continue	alet	aconst	avar
pass	with	comma	dotbetween	dot	lsb	rsb
in	type	newline	var	int	rcb	lcb
try	catch	throw	finally	scolon	switch	case
default	delete					

Production(P)

Production yang kami buat memiliki lebih dari 800 baris yang terdiri dari terminal dan nonterminal CFG yang akan diolah menjadi CNF.

Start Symbol(S)

Start symbol yang kami gunakan adalah **S**. Apabila grammar yang ditentukan bisa mencapai symbol **S** maka grammar tersebut diterima.

3.2 CNF Production

Chomsky Normal Form yang kami gunakan merupakan hasil dari fungsi yang kelompok kami buat yaitu CFGtoCNF sehingga kami mendapatkan file CNF yang sesuai dengan CFG kami. Namun, CNF yang kami hasilkan terlalu banyak untuk dimuat di laporan maka hasil dari CNF yang kami buat tidak akan ditampilkan pada laporan ini.

BAB IV
IMPLEMENTASI FUNGSI

4.1. File tokenizer.py

Tokenizer berisi fungsi dan prosedur untuk membaca file kemudian akan dibaca kata per kata yang ada di dalam file tersebut dan mengubahnya sebagai token. Token disini dapat dipecah menjadi 2 kategori yaitu terminal dan grammar.

no	Fungsi / Prosedur	Kegunaan
1	`matching`	Mencocokkan setiap karakter dengan token dan menyimpan token yang bersesuaian ke dalam array
2	`createToken`	Membaca file js atau txt dan mengubahnya menjadi token

4.2. File CFGtoCNF.py

CFGtoCNF berisi prosedur dan fungsi untuk mengubah bentuk CFG menjadi bentuk CNF. CFGtoCNF akan membaca grammar CFG dan mengkonversi sekaligus melakukan “write” pada CNF.

no	Fungsi/Prosedur	Kegunaan
1	`readFile`	Membaca file grammar
2	`addRule`	Menambah rule ke global dictionary(hash)
3	`convertGrammar`	Konversi CFG ke CNF
4	`grammarMapping`	Mapping CNF ke global dictionary
5	`writeGrammar`	Membuat file CNF.txt

4.3. File parser.py

Parser merupakan file yang berisi fungsi untuk melakukan pengecekan sintaks terhadap token yang sudah diolah oleh tokenizer dan akan dibandingkan dengan CNF yang didapat dari file CFGtoCNF.py. Parser ini akan memberikan output hasil *compile* dari file yang akan di cek.

No.	Fungsi/Prosedur	Kegunaan
1	`cykParse`	Parsing dari CNF menggunakan algoritma cyk. Parsing bernilai true saat CYK bisa mencapai Start Point(**S**)

4.4. File main.py

File main.py merupakan file yang berisikan program utama dari compiler node js ini. Main.py akan melakukan import terhadap semua file yang telah disebutkan sebelumnya. Program ini akan membaca sebuah file js lalu mengubahnya menjadi token. Lalu CFG yang telah disediakan akan diubah menjadi CNF lalu token yang telah didapat akan dibandingkan dengan CNF. File ini memiliki output berupa status keberhasilan compile dari program node.js tersebut.

BAB V

EKSPERIMEN

Dalam bagian ini kami melakukan beberapa uji coba menggunakan grammar yang diharuskan ada di spesifikasi. Berikut adalah beberapa kasus uji yang kami buat.

5.1. Conditional

Code:

```
let x = 0;
if(x == 0){
    console.log("x = 0");
}else if((x+1) == 2){
    console.log("Benar");
}else if((x+1) == 4){
    console.log("Mungkin");
}else{
    console.log("Salah");
}
```

Result:

```
Loading...
Checking your codes...
File name: test_IF_01.js

======================VERDICT========================

Selamat program kamu bisa di compile 🤣

=====================================================
```

Pada kondisi ini compile diterima karena kode sudah sesuai dengan grammar yang berlaku.

5.2 While Loop

Code:

```
var a = true

while (a){
    if (a == true){
        break
    }
}
```

Result:

```
Loading...
Checking your codes...
File name: test_LOOP_02.js

======================VERDICT========================

Selamat program kamu bisa di compile 🤣

=====================================================
```

Pada kondisi ini compile diterima karena kode sudah sesuai dengan grammer yang berlaku.

5.3 For Loop & Function

Code:

```
function simpleFor(x)
{
    function fun()
    {
        console.log("Gak guna");
    }
    for (i = 0; x < 2; x++)
    {
        x = i + 2;
        console.log(i + "iterasi");
    }
    return x;
}
```

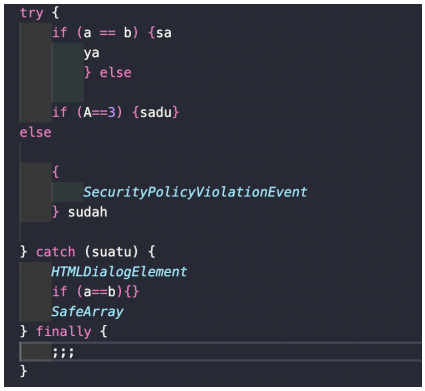
Result:

```
Loading...
Checking your codes...
File name: test_LOOP_01.js

======================VERDICT========================

Selamat program kamu bisa di compile 🤣

=====================================================
```

Pada kondisi ini compile diterima karena kode sudah sesuai dengan grammar yang berlaku.

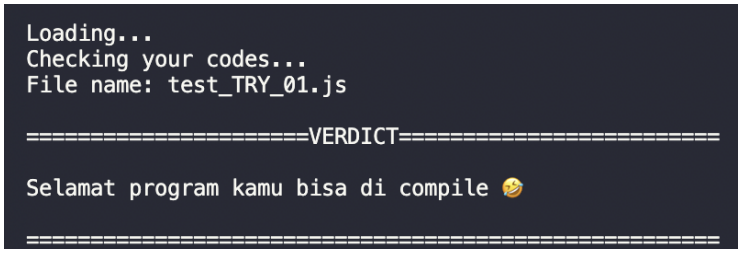
5.4. Try

Code:



```
try {
    if (a == b) {sa
        ya
        } else

    if (A==3) {sadu}
else

    {
        SecurityPolicyViolationEvent
    } sudah

} catch (suatu) {
    HTMLDialogElement
    if (a==b){}
    SafeArray
} finally {
    ;;;
}
```

Result:



```
Loading...
Checking your codes...
File name: test_TRY_01.js

=====================VERDICT=========================

Selamat program kamu bisa di compile 🤣

=====================================================
```

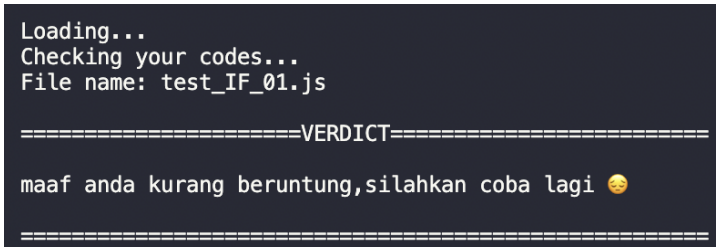
Pada kondisi ini compile diterima karena kode sudah sesuai dengan grammar yang berlaku.

5.5. Error

Code:

```
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";
    }
}
```

Result:



Pada kondisi ini compile ditolak karena kode tidak sesuai dengan grammar yang berlaku.

BAB VI
PENUTUP

6.1. Kesimpulan

Parser merupakan sebuah program yang berfungsi untuk mengurai. Dalam kasus ini yang akan diurai adalah sebuah program menggunakan node js. Parser ini sangat berguna untuk membuat sebuah compiler dan mengecek sintaks dari suatu program apakah sudah sesuai dengan grammarnya atau belum.

Berdasarkan tugas besar "Parser Bahasa Javascript(Node JS)", program yang kami buat berfungsi cukup baik mengingat adanya beberapa hal yang tidak dapat terhandle dengan baik. Testing yang dilakukan juga belum terlalu menyeluruh namun cukup untuk menjadi pondasi testing yang menurut kami layak. CFG yang telah kami buat juga belum sepenuhnya meliputi semua sintaks javascript karena javascript merupakan bahasa yang cukup asing bagi kami.

6.2. Saran

Dalam pengerjaan tugas besar kali ini, terdapat beberapa saran yang bisa kami utarakan untuk tugas besar ini.

- Manajemen waktu yang lebih baik sehingga masih dapat menyisihkan waktu untuk melakukan test error handling.
- Memulai pengerjaan tugas besar lebih awal sehingga selesai tidak mendekati deadline dan dapat mencoba lebih banyak testcase
- Memperbanyak referensi agar program yang dibuat bisa menjadi lebih efisien
- Membagi tugas dengan jelas agar pengerjaan menjadi lebih efisien
- Membuat branch untuk update agar tidak terjadi conflict satu sama lain

REFERENSI

1. https://www.tutorialspoint.com/automata_theory/context_free_grammar_introduction.htm
2. https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/
3. https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/
4. https://www.javatpoint.com/context-free-grammar
5. https://www.javatpoint.com/automata-chomskys-normal-form

LAMPIRAN

Link Repository Github : https://github.com/Raditss/TubesTBFO