IF2124 Teori Bahasa Formal dan Otomata

Laporan Tugas Besar

Compiler Bahasa Javascript

Kelompok TubesLagi

Tabitha Permalla – 13521111

Vanessa Rebecca Wiyono – 13521151

Brigita Tri Carolina – 13521156

PROGRAM STUDI INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

1. Dasar Teori

1.1 Teori Automata

Teori automata merupakan suatu cabang dari ilmu komputer, ditemukan pada abad ke-20, ketika para matematikawan sedang mengembangkan suatu mesin dengan fitur terbatas yang dapat melakukan kalkulasi dengan lebih cepat dan tepat.

Automata adalah sebuah model abstrak dari suatu mesin yang melakukan komputasi berdasarkan suatu input dengan cara berpindah *state* atau keadaan. Pada masing-masing *state,* suatu fungsi transisi akan menentukan konfigurasi selanjutnya. Hasilnya, ketika hasil komputasi telah mencapai konfigurasi *input* yang diterima, mesin akan menerima *input*. Automata yang cukup umum digunakan adalah mesin Turing.

Karakteristik sebuah mesin menurut teori automata di antaranya memiliki bagian-bagian sebagai berikut:

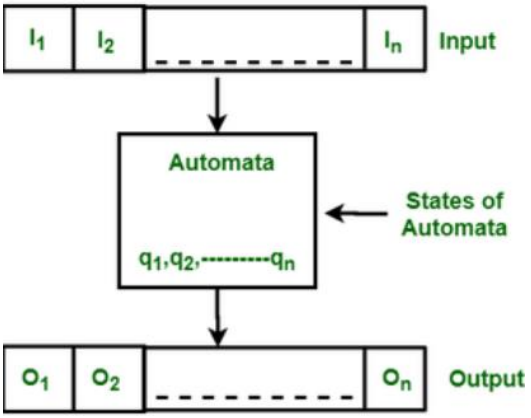
- *Input, input* menurut teori automata merupakan suatu urutan simbol dari sebuah set yang terbatas. Misalnya set l $\{x_1, x_2, x_3, \ldots, x_k\}$ dengan k adalah banyaknya *input*.
- *Outputs*, *outputs* menurut teori autamata merupakan suatu urutan simbol dari set yang terbatas. Misalnya set Z $\{y_1, y_2, y_3, \ldots, y_m\}$ dengan m adalah banyaknya *output.*
- *States*, suatu set Q, di mana keadaannya bergantung pada tipe automatanya.

Beberapa jenis automata di antaranya, diurutkan dari mesin yang paling sederhana ke mesin yang paling rumit:

- *Finite Automata*
- *Pushdown automata*
- *Linear-bounded automata*
- Turing *machine*

1.2 Finite Automata

Finite automata adalah suatu mesin sederhana yang membaca suatu *input* atau *pattern* untuk *regular language. Finite automata* atau *finite state machine* adalah abstrak dari mesin yang mempunyai lima elemen atau disebut juga *tuple. Finite automata* mempunyai suatu set dan *rules* yang mengatur perpindahan dari satu state ke state lain berdasarkan suatu input.



Gambar 1 Fitur Finite Automata

Fitur-fitur dari finite automata di antaranya:

Q : finite set of states

∑ : input symbols

q : initial states

F : set of final states

δ : transition function

1.3 Context Free Grammar (CFG)

Terdapat beberapa bahasa yang tidak regular yang tidak bisa di-*accept* oleh *finite automata* salah satunya *context-free-language.* CFG merupakan suatu mesin yang dapat menghandle *context-free-language.* Fitur-fitur CFG di antaranya G = (V, T, P, S) di mana:

- G adalah *grammar*, yang mengandung set dari *production rules*
- T adalah set terminal yang ditandai dengan huruf kecil
- V adalah set dari non-terminal simbol
- P adalah *set production rules*
- S adalah *start symbol*

The expressions are defined by the grammar

$$G = (\{E, I\}, T, P, E)$$

where $T = \{+, *, (,), a, b, 0, 1\}$ and P is the following set of productions:

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

Gambar 2 Contoh Context Free Grammar

1.4 Chomsky Normal Form (CNF)

CFG dikatakan merupakan sebuah Chomsky Normal Form jika setiap produksinya adalah salah satu dari kedua bentuk berikut:

1. A → BC (sisi sebelah kanan adalah dua variable)
2. A → a (sisi sebelah kanan adalah sebuah terminal

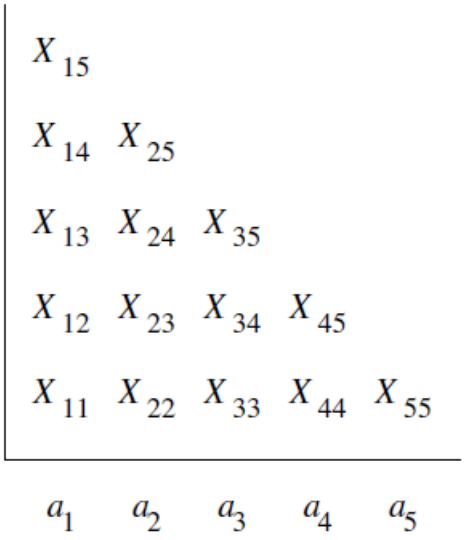
Teorema: Jika L adalah sebuah CFL, maka L – {ε} mempunyai CFG dalam bentuk CNF.

CNF harus memenuhi beberapa syarat di antaranya:

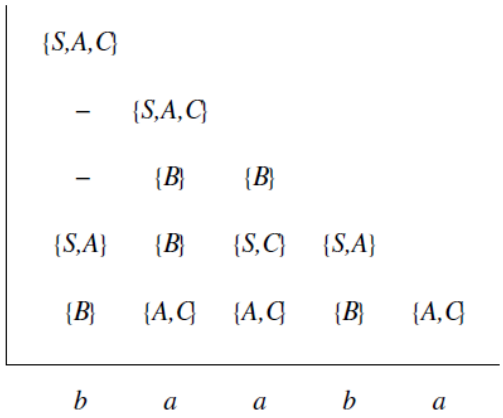
1. Tidak terdapat *useless variable*
2. Tidak terdapat *null production* (ε-*production*)
3. Tidak terdapat *unit production* (produksi ke satu *variable*)

Setiap grammar dalam CNF adalah CFG, dan setiap CFG dapat ditransformasikan ke bentuk CNF.

1.5 CYK



Gambar 3 Tabel Algoritma CYK



Gambar 4 Contoh CYK

Cocke-Younger-Kasami (CYK) adalah algoritma parsing yang memiliki efisiensi cukup tinggi untuk sebuah *context free grammar*. CYK dapat digunakan untuk menentukan apakah suatu bahasa merupakan bahasa dari CFG tertentu yang diberikan dalam bentuk CNF. Algoritma CYK dimulai dengan menuliskan kata *input* pada baris pertama dan menuliskan non-terminal simbol yang menurunkan simbol terminal input tersebut. Naik satu sel maka panjang dari simbol bertambah satu dan melakukan langkah sebelumnya yaitu mencari non-terminal yang menurunkan simbol dengan panjang lebih satu dari panjang sebelumnya. Langkah tersebut dilakukan berulang kali hingga panjang simbol sama dengan panjang simbol kata *input.* Langkah terakhir adalah untuk memeriksa apakah *start symbol* terdapat di baris terakhir. Jika iya, maka bahasa diterima oleh mesin, jika tidak maka bahasa tidak diterima oleh mesin.

2. CFG Production

Berikut adalah *context-free-grammar* yang telah dibuat untuk *compiler* bahasa Javascript.

Non-Terminal Symbol/Variable

S	SS	ENTER	PART	CLASS_STMT	FUNC_STMT
FOR_STMT	WHILE_STMT	SENTENCE	METHOD_STMT	WITH_STATE	IF_BLOCK
STRING	STRINGFORMAT	DICT	PARAMDICT	LIST	SET
ARIT_OP	ARIT_OPERATION	LOGI_VAR	LOGI_OP	LOGI_OPERATOR	PARAM
VAR_FUNC	STATIC	IF_STMT	ELIF_STMT	ARRAY	ASSIGN
ASSIGN_OP	OPERATION	ASSIGNMENT	ELIF_BLOCK	ARRAY_STMT	ARRAY_COMP
IDX	ARRAY	VAR	ALL	VARNUM	BOOL
VARNUM	EXP	VAR_FUNC	VAR_BANYAK	FUNC_SENTENCE	COMMENT_STMT
LOOP_SENTENCE	CASE_STMT	CASE_BYK	SWITCH_STMT	DEFAULT_STMT	TRY_BLOCK
TRY_STMT	CATCH_STMT	FINALLY_STMT	THROW_STMT	DELETE_STMT	

Terminal Symbol(T)

BREAK	CONST	CASE	CATCH	CONTINUE	DEFAULT
DELETE	ELSE	INCR	DECR	TRY	LET
NULL	SWITCH	THROW	FINALLY	EQUAL	ISEQ
KBKI	KBKA	TITIKKOMA	TITIKDUA	ADD	SUBTR
MUL	DIV	LE	L	GE	G
NEQ	SUBTREQ	MULEQ	SUMEQ	DIVEQ	AND
OR	NOT	IF	THEN	ELSE	WHILE
RANGE	FALSE	TRUE	NONE	CASE	CATCH
CLASS	FUNCTION	FOR	FROM	FORMAT	IMPORT
IN	IS	RETURN	PASS	WITH	COMMA
DOT	DOTBETWEEN	PETIKSATU	PETIKDUA	KSKI	KSKA
KKKA	KKKI	NUMBER	STRING	MULTILINE	VARIABLE
NEWLINE	TYPE	ARROW	SEMICOLON	COMMENT	COLON
ELIF					

3. Implementasi

3. 1 Struktur Data

File-file yang menyusun program ini di antaranya adalah cfgtocnf.py berisi algoritma untuk mengubah suatu cfg ke bentuk cnf, cykparser berisi algoritma cyk untuk mengecek apakah suatu input termasuk ke bahasa CFG tesebut, readgrammar berisi token dan pembacaan grammar dari file grammar.txt. File utama dari program ini adalah main.py berisi gabungan dan penggunaaan dari fungsi-fungsi pada file tadi untuk menentukan apakah suatu file.js memiliki struktur yang diterima atau jika tidak menghasilkan “syntax error”.

3.2 Fungsi dan Prosedur

3.2.1 Modul Lib cfgtocnf.py

Fungsi/Prosedur	Tujuan

CFG_to_CNF	Melakukan transformasi sebuah CFG ke CNF.

3.2.2 Modul Lib cykparser.py

Fungsi/Prosedur	**Tujuan**
cykAlgorithm	Melakukan parsing terhadap token yang diinput berdasarkan dictionary yang berisi CNF, parsing dilakukan dengan algoritma CYK. Bernilai true apabila *start symbol* berada pada token yang telah di-*input*.

3.2.3 Modul Lib readgrammar.py

Fungsi/Prosedur	**Tujuan**
read_grammar	Melakukan pembacaan file grammar.txt yang berisi CFG ke suatu variabel CFG.
tokenization, lexer	Membaca sebuah file js kemudian mengubahnya menjadi suatu token.
is_terminal	Fungsi boolean yang menghasilkan true jika suatu string adalah terminal, false jika tidak.
isVar	Menghasilkan true jika suatu string adalah variabel, false jika tidak.

4. Pengujian / Program Testing

Dalam bagian ini, kami telah melakukan beberapa kasus uji terhadap parser JavaScript yang telah kami buat. Berikut adalah beberapa kasus uji yang telah kami buat.

4. 1 Kasus Uji 1

Kata Kunci yang Diuji:	function, if, else, else if, return

Source Code JavaScript:

```
function do_something(x) {
  // This is a sample 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";
  }
}
```

Hasil : (token, isi tabel, dan verdict)

```
['FUNCTION', 'VARIABLE', 'KBKI', 'VARIABLE', 'KBKA', 'KKKI', 'NEWLINE', 'COMMENT', 'VARIABLE', 'IS', 'VARIABLE',
VARIABLE', 'VARIABLE', 'NEWLINE', 'IF', 'KBKI', 'VARIABLE', 'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEWLINE', 'RETURN', 'N
UM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELIF', 'KBKI', 'VARIABLE', 'ADD', 'NUM', 'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEW
LINE', 'IF', 'KBKI', 'TRUE', 'KBKA', 'KKKI', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELSE',
KKKI', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE', 'KKKA', 'ELIF', 'KBKI', 'VARIABLE',
'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELSE', 'KKKI', 'NEWLI
E', 'RETURN', 'STRING', 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE', 'KKKA', 'NEWLINE']
{'FUNCTION_STMT'}
ACCEPTED
```

4. 2 Kasus Uji 2

Kata Kunci yang Diuji:	for, let, if, continue

Source Code JavaScript:

```
for (let i = 0; i < 10; i++) {
  if (i == 3) {
    continue;
  }
}
```

Hasil : (token, isi tabel, dan verdict)

```
['FOR', 'KBKI', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'VARIABLE', 'L', 'NUM', 'SEMICOLON', 'VARIABLE',
'INCR', 'KBKA', 'KKKI', 'NEWLINE', 'IF', 'KBKI', 'VARIABLE', 'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEWLINE', 'CONTINUE'
 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE', 'KKKA']
{'LOOP_SENTENCE', 'S', 'FOR_STMT', 'SENTENCE', 'SS', 'FUNC_SENTENCE', 'START', 'PART'}
ACCEPTED
```

4. 3 Kasus Uji 3

Kata Kunci yang Diuji:	const, switch, case, var, break

Source Code JavaScript:

```
const x = 3;
switch(x) {
  case 1:
    var y = 1;
    break;
  case 2:
    var y = 2;
    break;
  default:
    var y = 3;
}
```

Hasil : (token, isi tabel, dan verdict)

```
['TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', 'SWITCH', 'KBKI', 'VARIABLE', 'KBKA', 'KKKI', 'NEWLIN
E', 'CASE', 'NUM', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', 'BREAK', 'SEMIC
OLON', 'NEWLINE', 'CASE', 'NUM', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', '
BREAK', 'SEMICOLON', 'NEWLINE', 'DEFAULT', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'N
EWLINE', 'KKKA']
{'FUNC_SENTENCE', 'SENTENCE', 'START', 'PART', 'SS', 'LOOP_SENTENCE', 'S'}
ACCEPTED
```

4. 4 Kasus Uji 4

Kata Kunci yang Diuji:	while

Source Code JavaScript:

```
while (i < 10) {
  x += i;
  i++;
}
```

Hasil : (token, isi tabel, dan verdict)

```
['TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', 'SWITCH', 'KBKI', 'VARIABLE', 'KBKA', 'KKKI', 'NEWLIN
E', 'CASE', 'NUM', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', 'BREAK', 'SEMIC
OLON', 'NEWLINE', 'CASE', 'NUM', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE', '
BREAK', 'SEMICOLON', 'NEWLINE', 'DEFAULT', 'COLON', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'N
EWLINE', 'KKKA']
{'FUNC_SENTENCE', 'SENTENCE', 'START', 'PART', 'SS', 'LOOP_SENTENCE', 'S'}
ACCEPTED
```

4. 4 Kasus Uji 5

Kata Kunci yang Diuji:	function, const, null, let, try, if, throw, catch, finally

Source Code JavaScript:

```
function myFunction() {
  const m = null;
  let x = value;
  try {
    if(x == 0){
      throw "is ZERO";
    }
  }
  catch(err) {
    x = 1;
  }
  finally {
    x = 2;
  }
}
```

Hasil : (token, isi tabel, dan verdict)

```
['FUNCTION', 'VARIABLE', 'KBKI', 'KBKA', 'KKKI', 'NEWLINE', 'TYPE', 'VARIABLE', 'EQUAL', 'NULL', 'SEMICOLON', 'NEW
LINE', 'TYPE', 'VARIABLE', 'EQUAL', 'VARIABLE', 'SEMICOLON', 'NEWLINE', 'TRY', 'KKKI', 'NEWLINE', 'IF', 'KBKI', 'V
ARIABLE', 'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEWLINE', 'THROW', 'STRING', 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE',
'KKKA', 'NEWLINE', 'CATCH', 'KBKI', 'VARIABLE', 'KBKA', 'KKKI', 'NEWLINE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON'
, 'NEWLINE', 'KKKA', 'NEWLINE', 'FINALLY', 'KKKI', 'NEWLINE', 'VARIABLE', 'EQUAL', 'NUM', 'SEMICOLON', 'NEWLINE',
'KKKA', 'NEWLINE', 'KKKA']
{'START', 'SS', 'FUNC_STMT', 'LOOP_SENTENCE', 'PART', 'FUNC_SENTENCE', 'SENTENCE', 'S'}
ACCEPTED
```

4. 6 Kasus Uji 6

Kata Kunci yang Diuji:	delete

Source Code JavaScript:

```
array = [1,2,3];
delete array[0];
```

Hasil : (token, isi tabel, dan verdict)

```
['VARIABLE', 'EQUAL', 'KSKI', 'NUM', 'COMMA', 'NUM', 'COMMA', 'NUM', 'KSKA', 'SEMICOLON', 'NEWLINE', 'DELETE', 'V
RIABLE', 'KSKI', 'NUM', 'KSKA', 'SEMICOLON']
{'SS', 'S', 'SENTENCE', 'PART', 'START', 'FUNC_SENTENCE', 'LOOP_SENTENCE'}
ACCEPTED
```

4. 7 Kasus Uji 7 (Unaccepted)

Source Code JavaScript:

```
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";
    }
  }
```

Hasil : (token, isi tabel, dan verdict)

```
['FUNCTION', 'VARIABLE', 'KBKI', 'VARIABLE', 'KBKA', 'KKKI', 'NEWLINE', 'COMMENT', 'VARIABLE', 'IS', 'VARIABLE', '
VARIABLE', 'VARIABLE', 'VARIABLE', 'NEWLINE', 'IF', 'KBKI', 'VARIABLE', 'ISEQ', 'NUM', 'KBKA', 'KKKI', 'NEWLINE',
'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELIF', 'VARIABLE', 'ADD', 'NUM', 'ISEQ', 'NUM', 'KKKI', 'NEWLINE
', 'IF', 'KBKI', 'TRUE', 'KBKA', 'KKKI', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELSE', 'KKKI
', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE', 'KKKA', 'ELIF', 'KBKI', 'VARIABLE', 'ISE
Q', 'NUM', 'KBKA', 'KKKI', 'NEWLINE', 'RETURN', 'NUM', 'SEMICOLON', 'NEWLINE', 'KKKA', 'ELSE', 'KKKI', 'NEWLINE',
'RETURN', 'STRING', 'SEMICOLON', 'NEWLINE', 'KKKA', 'NEWLINE', 'KKKA', 'NEWLINE']
set()
SYNTAX ERROR
```

5. Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan spesifikasi Tugas Pemrograman IF2124 Teori Bahasa Formal dan Otomata, program kami berjalan dengan baik dalam pengerjaan parser yang mengevaluasi sintaks bahasa pemrograman JavaScript (Node.js) untuk beberapa statement dan sintaks bawaan. Program kami belum mencakup seluruh sintaks dari Javascript sehingga tidak menutup kemungkinan untuk menghasilkan parsing dan evaluasi yang salah pada kasus-kasus lainnya. Namun, program kami dapat melakukan parsing dan evaluasi yang tepat untuk kasus testing serta mencakup kata kunci wajib pada grammar yang sebagaimana telah tertera pada spek.

5.2 Saran

Berdasarkan pengerjaan yang telah dilakukan, kami memiliki beberapa saran yang dapat diberikan terkait tugas besar ini.

1. Manajemen waktu yang lebih baik guna meningkatkan efektivitas dan efisiensi program
2. Perbanyak referensi mengenai Context-Free Grammar untuk menghasilkan algoritma yang lebih efektif

5. Repository

https://github.com/Bitha17/JS-Parser-by-TubesLagi

6. Pembagian Tugas

Nama – NIM	Pembagian Tugas
Tabitha Permalla – 13521111	Grammar, read grammar, tokenizer, main, testing
Vanessa Rebecca Wiyono – 13521151	Grammar, cyk parser, tokenizer (sebagian token exp), readme, laporan (kesimpulan)
Brigita Tri Carolina – 13521156	cfgtocnf, grammar, laporan (teori, cfg production, implementasi)

7. Referensi

https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html

https://www.geeksforgeeks.org/introduction-of-finite-automata/

https://www.javatpoint.com/automata-context-free-grammar

https://www.w3schools.com/js/default.asp