

Compiler Bahasa Python

Dibuat sebagai Tugas Besar

IF2124

Teori Bahasa Formal dan Automata



K-02

Kelompok 18

Louis Yanggara	13520063
Lyora Felicya	13520073
Averrous Saloom	13520100

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2021

Daftar Isi

BAB 1 TEORI DASAR.....	1
1.1 Finite Automata (FA).....	1
1.2 Context Free Grammar (CFG)	1
1.3 Cocke-Younger-Kasami (CYK)	4
1.4 Python.....	5
BAB 2 Hasil	6
2.1 Finite Automata.....	6
2.2 Context Free Grammar	7
2.3 Chomsky Normal Form	11
BAB 3 Implementasi dan Pengujian	12
3.1 Spesifikasi Teknik Program	12
3.1.1 File grammar_converter.py	12
3.1.2 File cnf_to_dict.py	13
3.1.3 File cyk_parser.py	13
3.1.4 File lexer.py	14
3.1.5 File main.py	14
3.2 Screenshot Hasil Pengujian	14
BAB 4 PENUTUP	19
4.1 Kesimpulan	19
REFERENSI.....	20

BAB 1 TEORI DASAR

1.1 Finite Automata (FA)

Finite Automata adalah sebuah mesin automata dari suatu bahasa regular. Finite Automata memiliki jumlah state yang banyaknya berhingga dan dapat berpindah-pindah dari suatu state ke state yang lainnya. Finite Automata dibagi menjadi *Deterministic Finite Automata*(DFA) dan *Non Deterministic Finite Automata*(NFA). Perbedaan utama dari DFA dan NFA adalah. Pada DFA, jika suatu state diberi inputan maka state tersebut akan selalu tepat menuju satu state, sedangkan pada NFA jika suatu state diberi inputan, maka mungkin saja bisa menuju beberapa state berikutnya.

1.2 Context Free Grammar (CFG)

Context Free Grammar atau CFG adalah tata bahasa formal di mana setiap aturan produksi adalah dalam bentuk $A \rightarrow B$ di mana A adalah pemroduksi dan B adalah hasil produksi. Batasannya hanyalah ruas kiri sebuah simbol variabel dan ruas kanan bisa berupa terminal, simbol, variabel ataupun ϵ . Dalam CFG, ada 4 komponen utama yaitu:

- Ada seperangkat simbol yang membentuk string bahasa yang didefinisikan atau yang biasa disebut sebagai *Terminal*
- Ada seperangkat variabel yang setiap variabel merepresentasikan sebuah *language* atau *set of string* yang biasa disebut *nonterminal*
- Ada sebuah variabel yang menandakan awal dari sebuah *language* yang biasa disebut sebagai *start symbol*
- Ada seperangkat aturan produksi yang merepresentasikan definisi suatu bahasa secara rekursif. Setiap aturan produksi terdiri dari:
 1. Sebuah variabel yang didefinisi oleh aturan produksi. Variabel ini sering disebut sebagai *head of production*.
 2. Simbol dari aturan produksi yaitu \rightarrow
 3. *String* dari nol atau lebih terminal dan variabel. *String* ini disebut sebagai *the body of the production* yang merepresentasikan salah satu cara untuk membentuk *string* pada *language* dari *head of production*

Misalkan G adalah sebuah CFG, kita dapat merepresentasikan G dengan keempat komponen yang sudah dijelaskan di atas yaitu $G=(V,T,P,S)$, dimana V adalah set dari variabel, T adalah terminal yang ada, P adalah aturan produksi, dan S adalah *start symbol*.

CFG perlu disederhanakan dengan tujuan untuk melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tidak perlu atau aturan produksi yang tidak berarti. Langkah-langkah untuk melakukan penyederhanaan CFG adalah:

- Eliminasi ϵ -production
- Eliminasi *unit production*
- Eliminasi simbol yang tidak diperlukan.

Bentuk CFG setelah disederhanakan biasa disebut sebagai *Chomsky Normal Form*(CNF) yang nantinya akan digunakan dalam algoritma *Cocke-Younger-Kasami* untuk melakukan parsing terhadap CFG.

Berikut ini adalah contoh penyederhanaan CFG:

$$A \rightarrow cAB \mid ab$$

$$B \rightarrow BaC \mid C$$

$$C \rightarrow bC \mid \epsilon$$

Langkah pertama : Eliminasi ϵ -production

- Hilangkan semua hasil produksi yang ϵ
- Jika $X \rightarrow \epsilon$, maka X adalah nullable
- Jika $Y \rightarrow X$, maka Y adalah nullable
- Jika $Z \rightarrow Xa$, maka $Z \rightarrow a$

$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid C \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

Langkah kedua : Eliminasi *unit production*

Jika ada hasil produksi yang terdiri dari 1 variable, maka hasil produksi tersebut disubstitusi dengan hasil produksi dari grammar dimana variable tersebut menjadi pemproduksi

$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid bC \mid b \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

Langkah ketiga : Eliminasi simbol yang tidak diperlukan

- **Uji Generate**

Jika $X \rightarrow YZ$, $Y \rightarrow aa$, $Z \rightarrow b$, maka X,Y,Z lolos uji generate

Jika $M \rightarrow PQ$, $P \rightarrow aa$, $Q \rightarrow QQ$, maka M dan Q tidak lolos uji generate

- **Uji Reachable**

Jika $S \rightarrow TU$ dimana S adalah start symbol maka T dan U lolos uji reachable

Hasil akhir :

$$A \rightarrow cAB \mid ab \mid cA$$

$$B \rightarrow BaC \mid bC \mid b \mid Ba \mid aC \mid a$$

$$C \rightarrow bC \mid b$$

1.3 Cocke-Younger-Kasami (CYK)

Cocke-Younger-Kasami algorithm atau yang biasa disebut sebagai CYK adalah sebuah algoritma untuk melakukan *parsing* terhadap sebuah CFG. Untuk menggunakan algoritma CYK, sebuah CFG harus diubah terlebih dahulu ke bentuk CNF. Algoritma CYK ini banyak digunakan karena memiliki efisiensi yang cukup tinggi untuk berbagai situasi. Berikut ini adalah salah satu contoh penggunaan algoritma CYK

Misalkan kita memiliki grammar :

$S \rightarrow NP VP$
 $VP \rightarrow VP PP$
 $VP \rightarrow V NP$
 $VP \rightarrow \text{eats}$
 $PP \rightarrow P NP$
 $NP \rightarrow \text{Det } N$
 $NP \rightarrow \text{she}$
 $V \rightarrow \text{eats}$
 $P \rightarrow \text{with}$
 $N \rightarrow \text{fish}$
 $N \rightarrow \text{fork}$
 $\text{Det} \rightarrow \text{a}$

Kemudian ada sebuah kalimat “she eats a fish with a for” yang akan dianalisis dengan menggunakan algoritma CYK. Untuk menganalisis gunakan tabel sebagai berikut:

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

Dari tabel diatas dapat dilihat bahwa pada bagian teratas tabel terdapat simbol start (S), maka kalimat tersebut dapat diproduksi oleh grammar.

1.4 Python

Python adalah bahasa *interpreter* tingkat tinggi (*high-level*), dan juga *general-purpose*. Python diciptakan oleh Guido van Rossum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan *code readability* dengan penggunaan *whitespace*-nya. Python adalah bahasa multiparadigma karena mengimplementasi paradigma fungsional, imperatif, berorientasi objek, dan reflektif.

Dalam bahasa Python *syntax* merupakan bagian yang sangat penting dan berpengaruh pada keberjalanan program. Berikut ini adalah beberapa hal yang harus diperhatikan dalam menuliskan bahasa Python

1. Comment: Pada bahasa Python comment terbagi menjadi 2 yaitu single line dan multiple line untuk single line, digunakan '#' sedangkan untuk multi-line digunakan ''' '''.
2. End-line: Pada bahas Python setiap statement atau pernyataan ditandai dengan baris baru.
3. Indentasi: Indentasi merupakan bagian yang sangat penting dalam pemrograman bahasa Python untuk menandakan suatu blok program atau logika tertentu. Namun pada Tugas Besar kali ini indentasi tidak akan diimplementasikan.
4. Tanda kurung: Pada bahasa Python, tanda kurung dimanfaatkan untuk mengelompokkan persamaan aritmatika maupun untuk memanggil sebuah fungsi.
5. Selain itu, dalam bahasa Python penggunaan "titik-dua" sangat penting pada fungsi-fungsi seperti if, elif, def, class, dan masih banyak lagi. Tidak adanya tanda titik dua menyebabkan terjadi *syntax error* dan program tidak dapat dijalankan.
6. Fungsi if, elif, dan else harus digunakan secara berurutan. Elif dan else harus didahului oleh if, apabila tidak maka akan menyebabkan terjadinya *syntax error*.
7. Penggunaan fungsi Def juga harus diikuti oleh nama fungsi tersebut, dan dilanjutkan dengan parameternya yang harus dituliskan di dalam tanda kurung, dan diakhiri dengan titik dua.

Untuk itu, akan dibuat sebuah compiler yang mengecek penggunaan *syntax* dalam bahasa Python meskipun masih belum dapat mengatasi semua *error* namun dapat digunakan untuk mengecek hal-hal dasar dalam melakukan pemrograman dengan bahasa Python.

BAB 2 Hasil

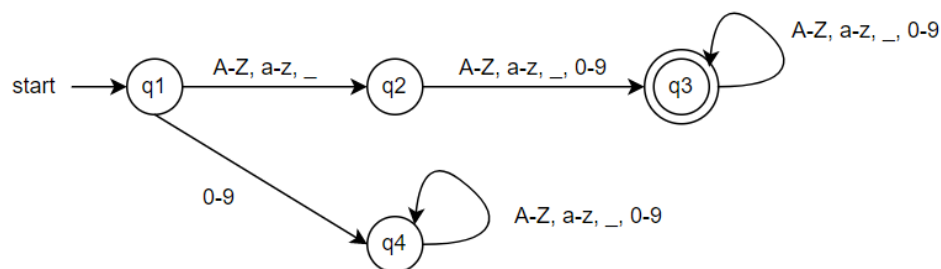
Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh programmer. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh 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.

Pada Tugas Besar IF2124, akan dibuat *compiler* untuk bahasa Python. Untuk membuat *compiler* tersebut, kami memanfaatkan *Cocke-Younger-Kasami*(CYK) . Pada dasarnya, CYK dapat digunakan ketika kita sudah memiliki suatu grammar dalam bentuk CNF, CNF dapat diperoleh dengan mengkonversi CFG terlebih dahulu.

Kata kunci bawaan Python yang harus ada adalah False, None, True, and, as, break, class, continue, def, elif, else, for, from, if, import, in, is, not, or, pass, raise, return, while, dan with.

2.1 Finite Automata

Dalam program ini, implementasi Finite Automata dilakukan untuk pengecekan suatu variabel. Variabel yang valid dalam python tidak boleh diawali dengan angka, dan hanya terdiri dari huruf A sampai Z, a sampai z, angka 0-9, serta karakter ‘_’ (*underscore*).



Finite Automata tersebut akan memeriksa karakter pertama dari nama variabel, apabila ditemukan angka, yaitu 0 – 9 maka akan masuk ke state q4 yang merupakan dead state. Jika masuk ke dead state, berarti nama variabel dalam program python tersebut tidak valid. Sebaliknya, jika tidak ditemukan angka, maka akan masuk ke state q2. Untuk karakter selanjutnya, semua valid sehingga akan lanjut ke state q3 yaitu accepting state.

2.2 Context Free Grammar

Dalam membuat suatu grammar, komponen paling dasar adalah Symbol, Terminal, Non-Terminal serta production rules. Semua CFG yang dibuat diletakkan dalam file `cfg.txt`. Berikut ini merupakan beberapa komponen penting dalam CFG :

1. Start Symbol

Kami menggunakan `S` sebagai simbol untuk menyatakan start symbol kami. `S` ini berfungsi untuk memutuskan apakah sebuah bentuk dapat diproduksi atau tidak. Start Symbol terdiri dari:

```
START → MULTI_COMMENT VARIABEL MULTI_COMMENT
START → COMMENT COMMENTIDN
START → IF_FINAL
START → ELIF_FINAL
START → ELSE_FINAL
START → FOR_FINAL
START → WHILE_FINAL
START → IMPORT_FINAL
START → FLOW_FINAL
START → PASS
START → CLASS_FINAL
START → WITH_FINAL
START → DEF_FINAL
START → EKSPRESI
```

Semua `START` ini merupakan fungsi yang akan diterima dalam grammar kami sesuai dengan yang ada di spesifikasi.

2. Terminal

Terminal merupakan kata-kata kunci paling mendasar yang nantinya akan digunakan sebagai *key* dalam program utama kami yang meliputi tanda baca, operator serta fungsi bawaan yang perlu diimplementasikan. Contohnya adalah `TRUE AND OR NOT FOR FROM IMPORT`, dan masih banyak lagi. Berikut ini merupakan daftar Terminal yang kami implementasikan:

```
IF → 'IF'
ELSE → 'ELSE'
ELIF → 'ELIF'
FALSE → 'FALSE'
```

NONE → 'NONE'
TRUE → 'TRUE'
AND → 'AND'
OR → 'OR'
NOT → 'NOT'
FOR → 'FOR'
IN → 'IN'
WHILE → 'WHILE'
FROM → 'FROM'
IMPORT → 'IMPORT'
AS → 'AS'
BREAK → 'BREAK'
RETURN → 'RETURN'
PASS → 'PASS'
RAISE → 'RAISE'
CONTINUE → 'CONTINUE'
CLASS → 'CLASS'
DEF → 'DEF'
IS → 'IS'
WITH → 'WITH'
QUOTE → 'QUOTE'
DQUOTE → 'DQUOTE'
COMPARE → 'COMPARE'
NUMBER → 'NUMBER'
VARIABLE → 'VARIABLE'
COMPARISON → 'COMPARISON'
ASSIGNMENT → 'ASSIGNMENT'
ARITMATIKA → 'ARITMATIKA'
COLON → 'COLON'
DOT → 'DOT'
COMMA → 'COMMA'
NEWLINE → 'NEWLINE'
KURUNG_BUKA → 'KURUNG_BUKA'
KURUNG_TUTUP → 'KURUNG_TUTUP'
KURUNG_SIKU_BUKA → 'KURUNG_SIKU_BUKA'
KURUNG_SIKU_TUTUP → 'KURUNG_SIKU_TUTUP'
COMMENT → 'COMMENT'
MULTI_COMMENT → 'MULTI_COMMENT'

Semua Terminal ini merupakan kata kunci yang akan dibuat rulesnya di dalam program utama kami untuk mengecek setiap kata kunci fungsi Python yang kami implementasikan. Bahasa yang digunakan cukup jelas sehingga tidak memerlukan penjelasan tambahan.

3. Non Terminal

Pada Non Terminal terdapat semua aturan produksi yang dapat dibentuk dalam bahasa Python. Untuk Non Terminal, kami membaginya menjadi beberapa bagian sesuai dengan fungsinya masing-masing yaitu:

CASE ARITMATIKA : Merupakan Daftar dari setiap kemungkinan dari kalimat yang menggunakan simbol aritmatika seperti 1+1 dan tes + 1

ARITMATIKA_NUMBER -> NUMBER

ARITMATIKA_NUMBER -> VARIABEL

ARITMATIKA_NUMBER -> ARITMATIKA_NUMBER ARITMATIKA_NUMBER

ARITMATIKA_EXPRS -> NUMBER ARITMATIKA ARITMATIKA_NUMBER

ARITMATIKA_EXPRS -> VARIABEL ARITMATIKA ARITMATIKA_NUMBER

MEMBUAT BASE CASE : Merupakan Setiap elemen yang mungkin terjadi untuk sebuah command pada Python seperti dalam penggunaan fungsi maupun looping

BASE -> VARIABEL

BASE -> VARIABEL BASE

BASE -> NUMBER

BASE -> TRUE

BASE -> FALSE

BASE -> NONE

BASE -> ARITMATIKA_EXPRS

BASE -> KURUNG_BUKA KURUNG_TUTUP

BASE -> KURUNG_SIKU_BUKA KURUNG_SIKU_TUTUP

BASE -> FUNC_SIKU DOT VARIABEL

BASE -> KURUNG_BUKA TESTLIST_COMP KURUNG_TUTUP

BASE -> KURUNG_SIKU_BUKA TESTLIST_COMP KURUNG_SIKU_TUTUP

BASE -> VARIABEL ARITMATIKA NUMBER ASSIGNMENT NUMBER

BASE -> QUOTE BASE QUOTE

BASE -> DQUOTE BASE DQUOTE

ARGUMENT -> VARIABEL

ARGUMENT -> ARGUMENT COMMA ARGUMENT

PARAMETER -> KURUNG_BUKA KURUNG_TUTUP

PARAMETER -> KURUNG_BUKA ARGUMENT KURUNG_TUTUP

PARAMETER_SIKU -> KURUNG_SIKU_BUKA NUMBER

KURUNG_SIKU_TUTUP

PARAMETER_SIKU -> KURUNG_SIKU_BUKA VARIABEL

KURUNG_SIKU_TUTUP

FUNC -> VARIABEL PARAMETER
FUNC_SIKU -> VARIABEL PARAMETER_SIKU

PERBANDINGAN :Merupakan daftar setiap kemungkinan dalam penggunaan operasi perbandingan seperti $x < 3$ atau $4 > 10$

COMPARE_BASE -> VARIABEL COMPAREX NUMBER
COMPARE_BASE -> ARITMATIKA_EXPRS COMPAREX BASE
COMPARE_BASE -> ARITMATIKA_EXPRS
COMPARE_BASE -> FUNC_SIKU
COMPARE_BASE -> FUNC
COMPARE_BASE -> FUNC_SIKU COMPAREX BASE
COMPARE_BASE -> FUNC COMPAREX BASE
COMPARE_BASE -> COMPARE_BASE COMPAREX COMPARE_BASE
COMPARE_BASE -> BASE
COMPAREX-> COMPARISON
COMPAREX -> IS
COMPAREX -> IS NOT
COMPAREX -> IN_FINAL
COMPAREX -> NOT IN_FINAL

EKSPRESI MATEMATIKA: berisi semua kemungkinan dalam ekspresi matematika yang melibatkan operator assignment seperti “==” atau “+=”

EKSPRESI -> VARIABEL ASSIGNMENT FUNC
EKSPRESI -> VARIABEL ASSIGNMENT FUNC_SIKU
EKSPRESI -> VARIABEL ASSIGNMENT NUMBER
EKSPRESI -> VARIABEL ASSIGNMENT VARIABEL
EKSPRESI -> ARITMATIKA_EXPRS ASSIGNMENT NUMBER

CEK CLASS DEF : berisi semua kemungkinan dalam pemanggilan def dan class

DEF_BASE -> DEF VARIABEL
DEF_FINAL -> DEF_BASE PARAMETER COLON
CLASS_BASE -> CLASS VARIABEL
CLASS_FINAL -> CLASS_BASE PARAMETER COLON

CEK FOR IN WHILE RAISE RETURN BREAK CONTINUE

FOR_BASE -> FOR VARIABEL
FOR_FINAL -> FOR_BASE IN_FINAL COLON
WHILE_FINAL -> WHILE TEST COLON
IN_FINAL -> IN VARIABEL
RETURN_STATEMENT -> RETURN
RETURN_STATEMENT -> RETURN VARIABEL
RETURN_STATEMENT -> RETURN NUMBER
RETURN_STATEMENT -> RETURN EKSPRESI
RETURN_STATEMENT -> RETURN BASE
TES_RETURN -> VARIABEL QUOTE
FROM_RAISE -> FROM_T TEST

```

RAISE_STATEMENT -> RAISE
RAISE_STATEMENT -> RAISE TEST
RAISE_STATEMENT -> RAISE TEST FROM_RAISE
FLOW_FINAL -> BREAK
FLOW_FINAL -> CONTINUE
FLOW_FINAL -> RETURN_STATEMENT
FLOW_FINAL -> RAISE_STATEMENT

# CEK FROM IMPORT AS
FROM_BASE -> FROM VARIABEL
IMPORT_BASE -> IMPORT VARIABEL
AS_BASE -> AS VARIABEL
IMPORT_FINAL -> IMPORT_BASE
IMPORT_FINAL -> IMPORT_BASE AS_BASE
IMPORT_FINAL -> FROM_BASE IMPORT_BASE
IMPORT_FINAL -> FROM_BASE IMPORT_BASE AS_BASE

# CEK IF ELIF ELSE
IF_FINAL -> IF TEST COLON
IF_FINAL -> IF EKSPRESI COLON
ELIF_FINAL -> ELIF TEST COLON
ELSE_FINAL -> ELSE COLON

# CEK WITH
WITH_BASE -> TEST
WITH_BASE -> TEST AS ARITMATIKA_EXPRS
WITH_BASE2 -> WITH_REC COLON
WITH_REC -> COMMA WITH_BASE
WITH_REC -> WITH_REC WITH_REC
WITH_FINAL -> WITH WITH_BASE COLON
WITH_FINAL -> WITH WITH_BASE WITH_BASE2

```

2.3 Chomsky Normal Form

CNF merupakan hasil konversi dari CFG sesuai dengan teori yang telah dijelaskan pada bagian sebelumnya. Untuk program kami, kami menggunakan algoritma converter yang sudah ada di internet untuk langsung menghasilkan CNF dari CFG. Hasil konversi disimpan dalam file cnf.txt.

BAB 3 Implementasi dan Pengujian

3.1 Spesifikasi Teknik Program

Program yang kami buat terdiri atas 5 file, yaitu *grammar_converter.py*, *cnf_to_dict.py*, *cyk_parser.py*, *lexer.py*, dan *main.py*.

3.1.1 File *grammar_converter.py*

File *grammar_converter.py* merupakan file yang berisi fungsi dan prosedur untuk mengubah Context Free Grammar (CFG) menjadi Chomsky Normal Form (CNF). Program ini diambil dari referensi <https://github.com/stefanbehr/cky>. Program dijalankan dengan perintah `python grammar_converter.py [file cfg] [file output]`. Hasil konversi CNF akan dituliskan dalam file output yang telah dituliskan pada perintah tersebut. Bentuk file input diasumsikan tertulis dengan LHS dan RHS yang dipisahkan tanda panah (\rightarrow), produksi yang banyak dipisahkan dengan (|), terminal dituliskan didalam tanda petik, dan comment diawali dengan '#' akan diabaikan.

Berikut adalah daftar fungsi/prosedur yang ada di dalam *grammar_converter.py*.

No.	Fungsi/Prosedur	Spesifikasi
1.	<code>find_terminals(rule)</code>	Mengembalikan list yang berisi terminal
2.	<code>stringify(rule)</code>	Menerima input rules CFG dan mengubahnya menjadi string.
3.	<code>add_rule(rule)</code>	Menambahkan rules yang telah dibuat ke dalam dictionary.
4.	<code>replace_terminals(rule,terminals,indices,i)</code>	Mengganti terminal yang ada ke dalam non-terminal, menambahkan rule baru dari rules sebelumnya.
5.	<code>replace_long(rule,i)</code>	Mengembalikan rule yang sudah dimodifikasi.
6.	<code>convert_long_and_hybrid(filename)</code>	Mengubah production yang panjang ke dalam bentuk CNF.
7.	<code>convert_unit(reserves,top)</code>	Menghilangkan unit production.
8.	<code>main</code>	Menerima input file CFG dan mengubahnya ke dalam CNF menggunakan prosedur-prosedur diatas. Hasilnya dituliskan dalam file output.

3.1.2 File `cnf_to_dict.py`

File ini berisi fungsi untuk mengonversi aturan *cnf* ke dalam *dictionary* karena spesifikasi implementasi yang kami buat.

No.	Fungsi/Prosedur	Spesifikasi
1.	<code>cnf_file_to_dict(file)</code>	Mengembalikan dictionary dari aturan-aturan CNF yang didefinisikan pada file
2.	<code>cnf_file_to_dict_indra(cfg_file)</code>	Mengembalikan dictionary dari aturan-aturan CNG yang didefinisikan pada file, tetapi menggunakan implementasi dari referensi. Hanya untuk eksperimen, tidak dipakai.

3.1.3 File `cyk_parser.py`

Secara umum, gambaran pendekatan algoritma cyk mengikuti prinsip yang dibahas pada video youtube https://www.youtube.com/watch?v=SFQ-owZaU_s. Prinsip-prinsip tersebut kemudian diimplementasi secara eksperimental. Setelah berbagai pendekatan implementasi, akhirnya didapatkan pendekatan yang paling benar.

List fungsi dan prosedur yang signifikan pada `cyk_parser.py`:

No.	Fungsi/Prosedur	Spesifikasi
1.	<code>cyk_parser(text,CNF)</code>	Menerima text dalam bentuk <i>array of string</i> yakni simbol-simbol yang didapatkan dari lexer, dan CNF dalam bentuk <i>dictionary of array</i> . Mengembalikan array berisi kebenaran grammar dan matriks cyk yang dihasilkan dari proses.
2.	<code>find_rightmost_left_of_unused_symbol(matrix, row, col)</code>	Menerima matriks cyk, kolom, serta baris. Mengembalikan nilai symbol non-null yang belum diproses di kiri elemen yang paling pertama
3.	<code>find_topmost_down_of_unused_symbol(matrix, row, col)</code>	Menerima matriks cyk, kolom, serta baris. Mengembalikan nilai symbol non-null yang belum diproses di kiri elemen yang paling pertama
4.	<code>is_grammar_fulfilled_in_cyk_matrix(start_state, cyk_matrix, CNF)</code>	Mengembalikan true jika pada ujung matrix cyk dapat dibentuk himpunan yang mengandung simbol start
5.	<code>nearest_non_terminal(CNF, input1, input2)</code>	Mencari parent terminal terdekat dari tiap pasangan input1 dan input2 atau mencari parent terminal terdekat dari tiap elemen input1
6.	<code>process_diagonal_matrix_from_</code>	Memproses diagonal submatriks matriks cyk. Sehingga berisi parent terminal terdekat dari

	right_to_left(CNF, matrix, diagonalKe)	simbol-simbol yang dapat diproses di diagonal sebelumnya.
7.	is_unused(matrix, row, col)	Mengembalikan true jika suatu simbol di row dan col belum dipakai untuk proses di sebelumnya, dicek dengan mengecek apakah elemen di atas dan di samping kanannya masih NULL atau tidak.
8.	fill_diagonal_matrix_with_nearest_non_terminal(CNF, matrix, text)	Mengisi diagonal matriks cyk. Digunakan karena penulis merasa pengisian diagonal matriks awal berbeda penanganannya dengan upamatriks.

3.1.4 File lexer.py

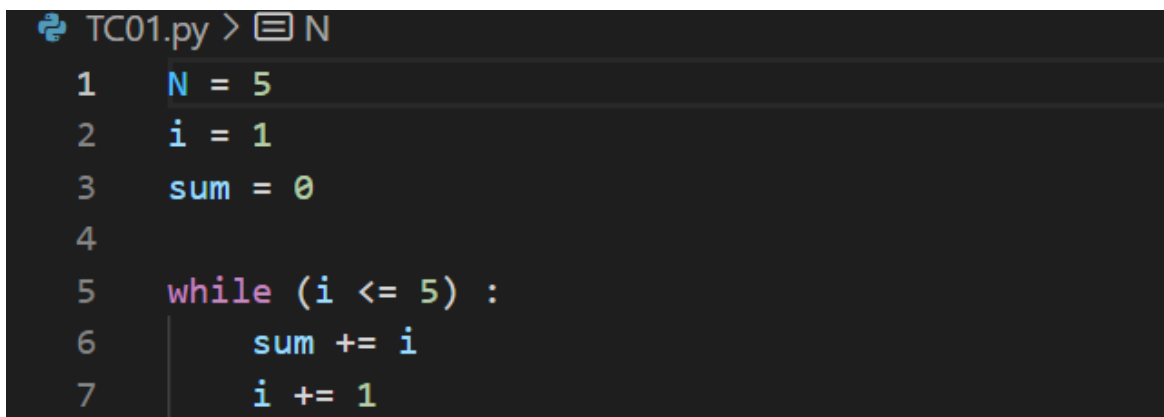
File lexer didapatkan dari referensi <https://github.com/indraf Nugroho/Python-Compiler/blob/master/modified/lexer.py>, yang ternyata juga direferensi dari Eli Bendersky yang membuatnya pada tahun 2010.

3.1.5 File main.py

File main.py berisi source code yang merupakan main program dari program compiler python yang kami buat. Program ini mengimport file lexer, file cnf_to_dict untuk mengubah CNF ke dalam bentuk dictionary Python, serta cyk_parser untuk melakukan proses parsing terhadap dictionary Python yang sudah dikonversi sebelumnya. File ini juga berisi rules yang akan digunakan pada program lexer.

3.2 Screenshot Hasil Pengujian

TC01.py



```

TC01.py > N
1  N = 5
2  i = 1
3  sum = 0
4
5  while (i <= 5) :
6      sum += i
7      i += 1

```



```
Masukkan nama file: TC01.py
line 1: Accepted
line 2: Accepted
line 3: Accepted
line 4: Accepted
line 5: Accepted
line 6: Accepted
line 7: Accepted
```

Pada file ini dapat dilihat seluruh line benar karena memenuhi aturan python.

TC02.py

```
if(true):
    aas+=1
elif(true == true):
    print(true)
if(a > 4):
    a = 3
elif(a < 4 and a > 1):
    a-=9
else:
    a-=1
```

```
Masukkan nama file: TC02.py
line 1: Accepted
line 2: Accepted
line 3: Accepted
line 4: Accepted
line 5: Accepted
line 6: Accepted
line 7: Accepted
line 8: Accepted
line 9: Accepted
line 10: Accepted
line 11: Accepted
```

Pada TC02 ini dites untuk melihat apakah bisa mengevaluasi syntax if , elif dan else dengan benar. Terlihat bahwa secara syntax program TC02.py ini benar, dan hasil parsing kami juga menunjukkan bahwa seluruh baris itu **Accepted**.

TC03.py

```
TC03.py > do_something
1  def do_something(x):
2      if x == 0 + 1
3          return 0
4      elif x + 4 == 1:
5          else:
6              return 2
7      elif x == 32:
8          return 4
9      else:
10         return Doodoo
```

Masukkan nama file: TC03.py
line 1: Accepted
line 2: Syntax error
line 3: Accepted
line 4: Accepted
line 5: Accepted
line 6: Accepted
line 7: Accepted
line 8: Accepted
line 9: Accepted
line 10: Accepted

Pada TC03.py ini terlihat bahwa terjadi Syntax Error pada baris ke 2, karena tidak ada tanda titik dua setelah penggunaan if.

TC04.py

```
TC04.py > do_something
1  def do_something(x):
2      # ini comment
3      x + 5 = 11
4      5 + 5 = 10
5      tes_prog = tes
6      if x == 0:
7          return 0
8      elif x + 4 == 1:
9          if True:
10             return 3
11         else:
12             return 2
13     elif x == 32:
14         return 4
15     else:
16         return "tes"
17
```

```
Masukkan nama file: TC04.py
line 1: Accepted
line 2: Accepted
line 3: Syntax error
line 4: Syntax error
line 5: Accepted
line 6: Accepted
line 7: Accepted
line 8: Accepted
line 9: Accepted
line 10: Accepted
line 11: Accepted
line 12: Accepted
line 13: Accepted
line 14: Accepted
line 15: Accepted
line 16: Accepted
line 17: Accepted
```

TC05.py

```
TC05.py > ...
1  def do_something(x):
2      ''' This is a sample multiline comment
3      '''
4      x + 2 = 3
5      if x == 0 + 1
6          return 0
7      elif x + 4 == 1:
8          else:
9              return 2
10     elif x == 32:
11         return 4
12     else:
13         return "Doodoo"
```

Masukkan nama file: TC05.py
line 1: Accepted
line 2: Accepted
line 3: Accepted
line 4: Syntax error
line 5: Syntax error
line 6: Accepted
line 7: Accepted
line 8: Accepted
line 9: Accepted
line 10: Accepted
line 11: Accepted
line 12: Accepted
line 13: Accepted

Pada TC05.py ini terlihat bahwa hasil parsing menunjukkan Syntax Error pada line 4 dan line 5. Ini benar karena syntax pada kedua baris tersebut tidak valid. Pada line 4, seharusnya fungsi if harus diakhiri dengan tanda titik dua setelah parameternya.

BAB 4 PENUTUP

4.1 Kesimpulan

Melalui Tugas Besar yang berjudul “Compiler Bahasa Python” ini kami telah membuat program yang dapat mengevaluasi syntax bahasa Python. Program ini sudah dapat mengevaluasi program-program sederhana dan mengeluarkan hasil yang tepat. Namun, program kami masih memiliki kekurangan karena terdapat kasus-kasus yang belum berhasil kami tangani seperti kasus multiline comment dan logika if(elif dan else harus dideklarasikan setelah adanya if). Hal itu disebabkan tidak tertanganinya newline pada program kami sehingga program tidak dapat mendeteksi command yang sudah ada sebelumnya.

Link repository github : <https://github.com/louisyyyy/Tubes-IF2124-TBFO>

Pembagian Tugas :

Nama	NIM	Tugas
Louis Yanggara	13520063	Membuat CFG, laporan
Lyora Felicya	13520073	Membuat CNF, laporan
Averrous Saloom	13520100	CYK parser, laporan

REFERENSI

<https://data36.com/python-syntax-essentials-and-best-practices/>

<https://jakevdp.github.io/WhirlwindTourOfPython/02-basic-python-syntax.html>

<https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/>

<https://www.geeksforgeeks.org/with-statement-in-python/>