

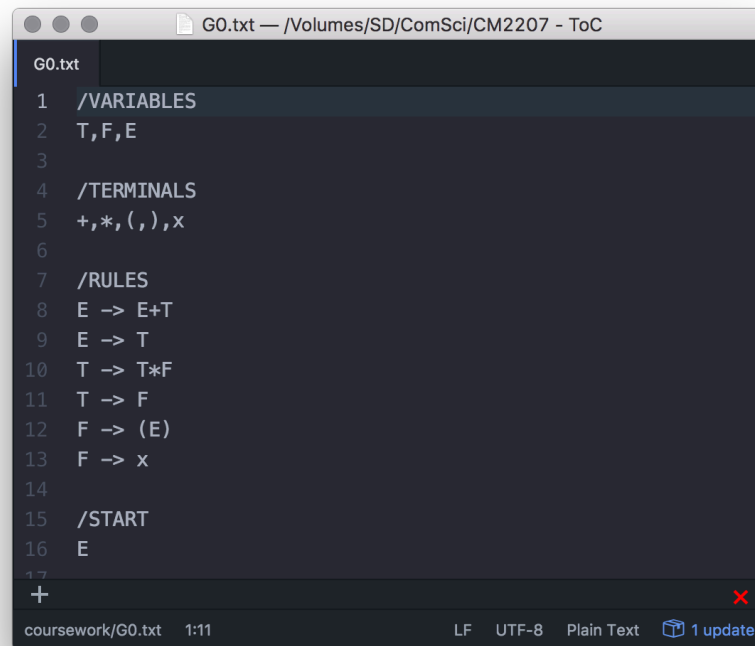
Theory of Computation

Coursework
C1527380

Table of Contents

Task 1.....	3
Task 2.....	5
Task 3.....	6
Further investigations	7
Bibliography	8

Task 1



```
G0.txt
1 /VARIABLES
2 T,F,E
3
4 /TERMINALS
5 +,*,(,),x
6
7 /RULES
8 E -> E+T
9 E -> T
10 T -> T*F
11 T -> F
12 F -> (E)
13 F -> x
14
15 /START
16 E
```

The screenshot shows a text editor window titled 'G0.txt' with a dark background. The file content is as follows:

```
1 /VARIABLES
2 T,F,E
3
4 /TERMINALS
5 +,*,(,),x
6
7 /RULES
8 E -> E+T
9 E -> T
10 T -> T*F
11 T -> F
12 F -> (E)
13 F -> x
14
15 /START
16 E
```

The editor interface includes a status bar at the bottom showing 'coursework/G0.txt 1:11', 'LF UTF-8 Plain Text', and '1 update'.

Figure 1: CFG G_0 text file

Figure 1 describes my encoding of the CFG G_0 .

A key aim of my encoding was to maximise readability; below I explain each part of my encoding:

- **Sections:**
 - Each tuple is separated into their respective sections, each section is headed by a "/" followed by the type of tuples contained below e.g. /VARIABLES
 - The "/" is there for clarity in differentiating the heading from the tuple content
 - The section name follows the "/" in capital letters, again to differentiate from the content below.
 - All the declarations must be written below the headings and must follow the respective rules for each.
- **Variable declarations:**
 - Variables are declared on a single line, separated by commas
 - They must be one character long
 - Whitespace must be denoted with and underscore: "_"
- **Terminal declarations:**
 - Terminals are also declared on a single line, separated by commas
 - They must be one character long
 - Whitespace must be denoted with and underscore: "_"
- **Rule declarations:**
 - Each rule is declared on a separate line
 - The rule's variable and production is separated with "->"

- The spaces either side of “->” are optional, but I recommend adding them for clarity
- **Start variable**
 - The start variable must be a single character and be a character from the set of variables
- **Empty string**
 - If any of the rules contain the empty string “ ϵ ”, add “ ϵ ” to the list of terminals so that the program knows that it is going to have to be aware of empty strings.

Task 2

For this task, I have written a program which takes a .txt file, with the encoding I defined in task 1, as a command line argument (see Figure 2). When run, my program reads the file containing the CFG and extracts the Variables, Terminals, Rules and Start variable – placing each into their respective data structure in my Python representation of a CFG.

```
ojs-[coursework]> python cfg.py G0.txt
```

Figure 2: Running the program in command line

My program then performs a sequence of operations shown in Figure 3, which converts the CFG G_0 into a CFG in Chomsky Normal Form (CNF). Where appropriate, I print to the terminal the current state of the CFG at each stage of the conversion.

As you can see, the final printed section displays the CFG in valid CNF:

- $A \rightarrow BC$
- $A \rightarrow a$
- $S \rightarrow e$

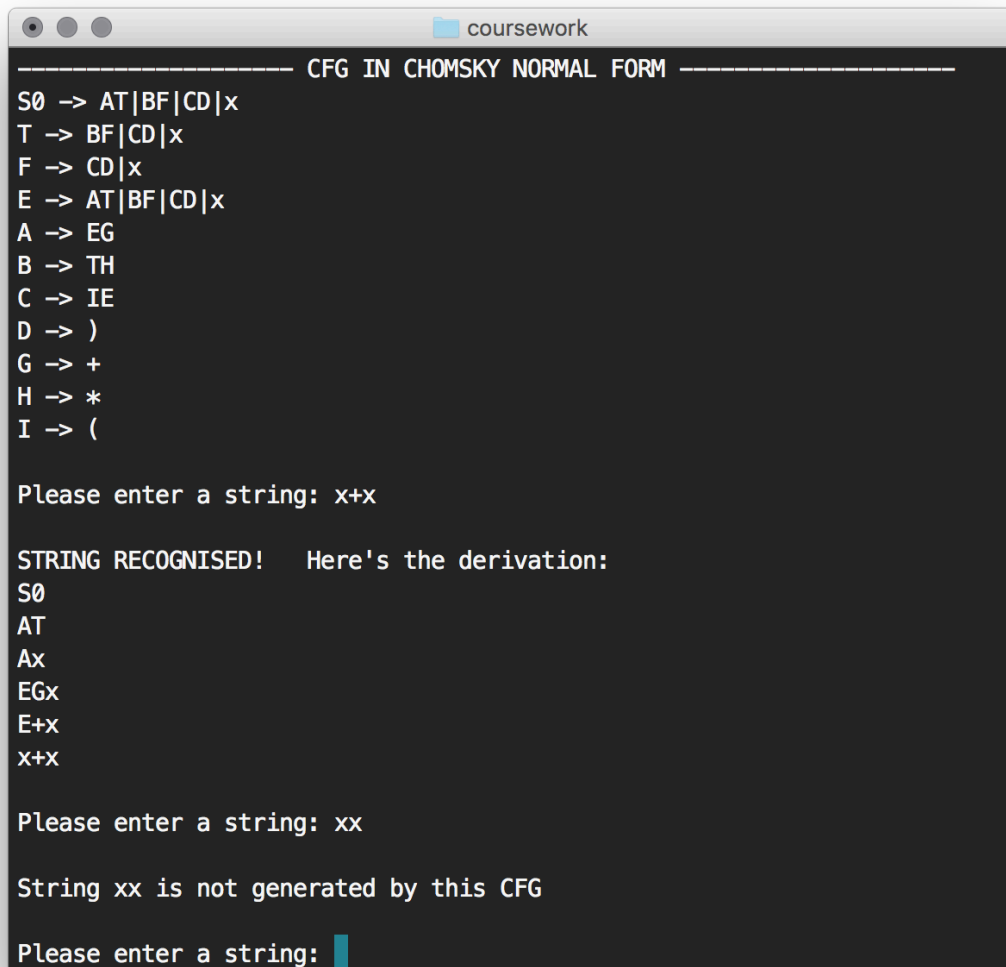
```
----- REMOVE UNIT RULES -----
Variables:      ['T', 'F', 'E']
Terminals:      ['+', '*', '(', ')', 'x']
Rules:          ['E -> E+T', 'E -> T', 'T -> T*F', 'T -> F', 'F -> (E)', 'F -> x']
Start:          E
----- CREATE NEW START VARIABLE -----
S0 -> E+T|T
T -> T*F|F
F -> (E)|x
E -> E+T|T
----- REMOVE UNIT RULES -----
S0 -> E+T|T*F|(E)|x
T -> T*F|(E)|x
F -> (E)|x
E -> E+T|T*F|(E)|x
----- REMOVE EMPTY SET RULES -----
S0 -> E+T|T*F|(E)|x
T -> T*F|(E)|x
F -> (E)|x
E -> E+T|T*F|(E)|x
----- CONVERT REMAINING RULES -----
S0 -> AT|BF|CD|x
T -> BF|CD|x
F -> CD|x
E -> AT|BF|CD|x
A -> EG
B -> TH
C -> IE
D -> )
G -> +
H -> *
I -> (
----- CFG IN CHOMSKY NORMAL FORM -----
S0 -> AT|BF|CD|x
T -> BF|CD|x
F -> CD|x
E -> AT|BF|CD|x
A -> EG
B -> TH
C -> IE
D -> )
G -> +
H -> *
I -> (
```

Figure 3: G_0 converted to CNF

Task 3

When my program is run, it outputs the CFG in CNF and then prompts the user for a string. The user enters a string, the program checks if the CFG can derive this string then displays the result to the user; if it can be derived, the sequence of derivation from the start variable "S0" is shown. If not, the program will notify that the string is not recognised. The program will repeatedly ask for input until CTRL+C is used to quit (to save you from having to restart the program each time 😊).

Figure 4 shows that G_0 accepts the string "x+x" but rejects the string "xx".



```
----- CFG IN CHOMSKY NORMAL FORM -----
S0 -> AT|BF|CD|x
T -> BF|CD|x
F -> CD|x
E -> AT|BF|CD|x
A -> EG
B -> TH
C -> IE
D -> )
G -> +
H -> *
I -> (

Please enter a string: x+x

STRING RECOGNISED! Here's the derivation:
S0
AT
Ax
EGx
E+x
x+x

Please enter a string: xx

String xx is not generated by this CFG

Please enter a string: █
```

Figure 4: String parsing for G_0

Time Complexity

As instructed by the coursework specification, I essentially constructed a derivation tree with a maximum depth of $2n-1$, and if any of the derivations matched the input string, the string is accepted.

The issue with this method is that it runs in exponential time k^n (where $n=|w|$, and w is the input string) because of repeated visits to nodes on the tree; this significantly slows down the parser for longer input strings. A better approach would be to use the CYK algorithm which uses Dynamic Programming to store the intermediate results so that the parser can operate in polynomial time (Cocke, *n.d.*). This reduces the time complexity from $O(k^n)$ to $O(n^3)$, a huge saving!

Ambiguity

If a rule has multiple productions, it can be considered ambiguous. This means that for a given string, it can be produced by multiple derivations. For example, if we attempt to parse “x+x” with the CFG G_0 in CNF, it can be derived in the following ways:

CFG G_0	Derivations for “x+x”
$S \rightarrow AT \mid BF \mid CD \mid x$	$(START)S \rightarrow AT \rightarrow EGT \rightarrow EGx \rightarrow E+x \rightarrow x+x$
$T \rightarrow BF \mid CD \mid x$	$(START)S \rightarrow AT \rightarrow Ax \rightarrow EGx \rightarrow E+x \rightarrow x+x$
$F \rightarrow CD \mid x$	
$E \rightarrow AT \mid BF \mid CD \mid x$	
$A \rightarrow EG$	
$B \rightarrow TH$	
$C \rightarrow IE$	
$D \rightarrow)$	
$G \rightarrow +$	
$H \rightarrow *$	
$I \rightarrow ($	

Both the derivations are valid, but my program will only display the most recently found derivation (easily changeable but not required in the scope of this coursework). A possible way to save time would be to halt the program upon finding the first derivation, this can be easily implemented but the program will still be running with exponential time complexity.

Any problems please email me: CoplestonO@cardiff.ac.uk

Bibliography

Cocke, Y. K. (n.d.). *CYK algorithm*. Retrieved from Wikipedia:
https://en.wikipedia.org/wiki/CYK_algorithm