ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 3: Generating Languages from Grammers

Yang Heechan

21800436@handong.edu

1. Introduction

For a string to be in a set of strings, this set must contain a rule that allows only certain string to become a subset or a member of the set. This rule can be defined recursively. And by this rule, strings that are a subset of a certain set can be found.

For this assignment, a C program (langdump.c) had to be written in which this program receives a .txt file that holds a recursively defined grammar and an integer value that limits the maximum length for a member string.

This C program uses this input arguments and displays all the member strings of a set in which the rule of this set is defined recursively by the .txt file.

1.1 Validation

The integer value to be received from the second argument of the program must be a value between 1 to 64. This value is maximum length that a member string must follow to become a member of the set. If this condition doesn't hold true, the program prints out an error message followed by the termination of the program.

The .txt file holding the recursively defined rule for a set of strings differentiates the rule by '::='. The left side of this string writes a symbol in which is a key for a sequence held on the right side of '::='. The length of the sequence placed on the right side can be up to 64 characters. The symbol 'S' holds a value to define another symbol which can be a case-sensitive alphanumeric word with a length of up to 16 characters. And by the assignment description, only up to 32 symbols can be defined.

2. Approach

2.1 Accumulating Data Input

Reading a text file that defines the grammar of a set of strings is saved in two data structures.

The first data structure (symbols) which is a one-dimensional array of strings that holds all the defined symbol.

The second data structure (sequences) which is a two-dimensional array of strings that holds a key and value. The key is a symbol that guides to a value which is the string sequence.

2.2 Recursive Function

2.2.1 Case 1

The first case to be in the recursive function is the case where the length of the current member string is greater than the given maximum length.

This stops the recursive function to derive in deeper on the given sequences of string.0000

2.2.1 Case 2

The second case in the recursive function is the case where the length of the current member string is less than the given maximum length and the sequence contains no other symbols.

A function, symbol_exist(), function is defined to check for the existence of a symbol in the current sequence. This function runs a loop by the symbols saved in the symbol data structure.

When both this condition satisfies, the sequence is classified to be a member string of the given set. Therefore, this sequence of string is printed out to the standard output.

2.2.1 Case 3

The third case in the recursive function is the case where all the cases above doesn't satisfy. Therefore, the length of the current sequence is within the maximum length and this sequence contains a symbol.

Since it contains a symbol, this symbol is replaced with another sequence that is defined to have the according key. Then the modified goes through the recursive function ready to be tested with these three cases.

So forth, this case in the recursive function goes deeper within the recursive grammar.

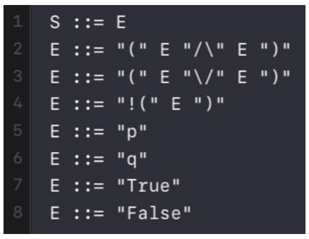
3. Evaluation

This section displays a real test case ran through program of langdump.

Two test cases will be evaluated: Propositional Logic Binary Palindrome. Each cases have its own text file defining the grammar.

The task that will be shown below is proved to be successful because the printed member strings to each case follow the grammar define in each text file. Secondly, all the printed member strings follow the rule of having a length not exceeding the maximum length given as the input.

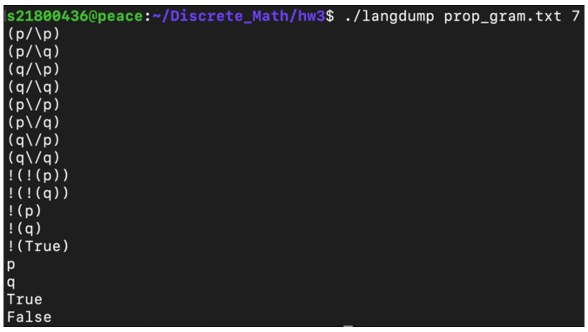
3.1 Propositional Logic



```
S ::= E
E ::= "(" E "/\" E ")"
E ::= "(" E "\/" E ")"
E ::= "!(" E ")"
E ::= "p"
E ::= "q"
E ::= "True"
E ::= "False"
```

Figure 3.1.1 Propositional Logic Grammar

Figure 3.1.1 shows the recursively define grammar for the propositional logic. Langdump program will read this file and accumulate all the data in the according data structure as explain in section 2.1.

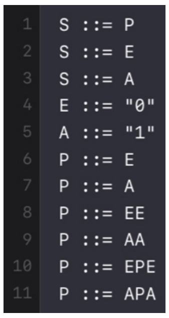


```
s21800436@peace:~/Discrete_Math/hw3$ ./langdump prop_gram.txt 7
(p/\p)
(p/\q)
(q/\p)
(q/\q)
(p\/p)
(p\/q)
(q\/p)
(q\/q)
!(!(p))
!(!(q))
!(p)
!(q)
!(True)
p
q
True
False
```

Figure 3.1.2 Output Result of Propositional Logic

Figure 3.1.2 shows the result of program that ran through the text file 'prop_gram.txt' (grammar text file for propositional logic) and a maximum string length of 7.

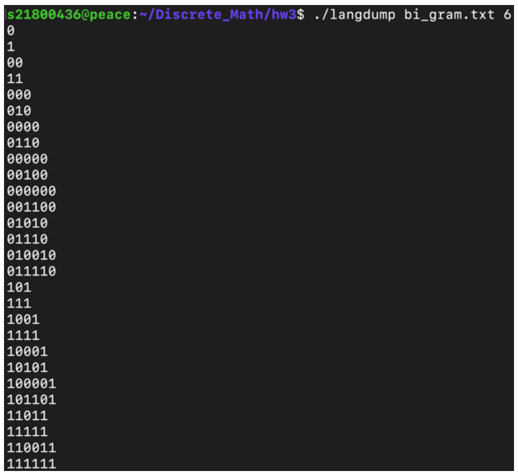
3.2 Binary Palindrome



```
S ::= P
S ::= E
S ::= A
E ::= "0"
A ::= "1"
P ::= E
P ::= A
P ::= EE
P ::= AA
P ::= EPE
P ::= APA
```

Figure 3.2.1 Binary Palindrome Grammar

Figure 3.2.1 shows the recursively define grammar for the binary palindrome. Langdump program will read this file and accumulate all the data in the according data structure as explain in section 2.1.



```
s21800436@peace:~/Discrete_Math/hw3$ ./langdump bi_gram.txt 6
0
1
00
11
000
010
0000
0110
00000
00100
000000
001100
01010
01110
010010
011110
101
111
1001
1111
10001
10101
100001
101101
11011
11111
110011
111111
```

Figure 3.2.2 Output Result of Binary Palindrome

Figure 3.2.2 shows the result of program that ran through the text file 'bi_gram.txt' (grammar text file for binary palindrome) and a maximum string length of 6.

4. Discussion

A recursive algorithm as a function was implemented to apply the grammar rule as deepest possible. Since the sequences have been structured in a two-dimensional string array, a for loop have been used within the recursive structure to check each sequence cases to each current sequence when it contains a defined symbol. However, I believe that this could have been implemented without a for loop but by giving the next symbol/sequence map to be tested.

The text file for recursively defined grammar for parenthesis and positive arithmetic expression was not made. Defining something recursively grew to become complicated as of thinking of many different cases for the grammar to satisfy. Therefore, I have learned to not think by head but also draw by hand to form a recursive definition.

5. Conclusion

In conclusion, the given assignment finds all the member strings that satisfies to a certain set of string by following the given rule defined recursively in the .txt file and the maximum length in characters.

The input data have been stored in appropriate data structures and have been well accessed in the recursive function to grow deeper in the recursive string sequence and determine the case of string to be a member string. By symbol and sequence the cases are well test print or replace the given string sequence.