ITP 20002-03 Discrete Mathematics, Fall 2021

Homework 3. Generating Languages from Grammars

Shin Hong

hongshin@handong.edu

1. Introduction

A *language* is a set of strings that follow certain string construction rules. We call such rules the *grammar* of the language. In this homework, you are asked to write a C program that receives a grammar description and then enumerates all member strings up to a specified length. The grammar of most computer languages has recursive aspects to specify recurring patterns in a string. Given such recursive grammar rules, your program must use a recursion to generate syntactically valid strings.

This homework description specifies how grammar rules are expressed in an input file (i.e., the grammar of grammars) (Section 2) and how they should be interpreted (Section 3). In addition, this homework gives you three questions for you to find the answers with your own program (Section 4). You should submit all results including your implementation and your report by **11:59 PM, 17 November (Wed)**. Each student should accomplish all tasks of this homework individually.

2. Expressing Grammar

The grammar of a certain language can be expressed as *derivation rules* each of which specifies how a symbol can be replaced with a finite sequence of symbols and/or strings[1]. Every member of a language should be the result of applying a series of derivations from a starting symbol. For example, the grammar of propositional logic with two propositional variables can be written as the following derivation rules:

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

This grammar consists of eight derivation rules. There are two symbols `S` and `E`, and one or multiple derivations rules are given to specify how the symbol at the left-hand side can be rewritten as the sequence in the right-hand side. Symbol `S` is the starting symbol from which all member strings are derived. Among multiple rules with the same left-hand side, any one of them can be applied for derivation. Note that the second to forth rules are defined recursively to express that the string of an expression embeds another expression.

Once all symbols are replaced, the concatenation of the concrete strings is identified as a member of the language. For example, we can find the "`((p/\True)\/q)`" belongs to the given language for the following derivation series:

```
S ⇒ E ⇒ (E\/E) ⇒ (E\/q) ⇒ ((E/\E)\/q)
⇒ ((p/\E)\/q) ⇒ ((p/\True)\/q)
```

One way to recognize whether or not a given string is of a given grammar is to explore all possible derivation sequences up to the length of the given string and check if any result is the same as the given string. Note that there may be multiple derivation sequences for generating the same string.

3. Language Generation Program

You are asked to construct `langdump`, a general-purpose language generation program which enumerates all member strings up to a certain length for a given grammar description. This type of programs is used in practice for constructing test cases of a program receiving structured inputs. A command for `langdump` should be as follows:

`$./langdump` <*grammar-description-file*> <*max-length*>

A grammar description file is a text file where each line represents a derivation rule. [2] Each line starts with a symbol followed by the definition operator sign ("`::=`") and a finite list of up to 32 symbols and/or strings. A symbol is a case-sensitive alphanumeric word with no more than 16 characters. A string is a finite sequence of zero or up to 64 characters surrounded by the double quotation marks. There must be one or more whitespaces between two adjacent items in a derivation rule description. In a grammar description file, there must be at least one derivation rule that defines the starting symbol `S`. Assume that a grammar description file does not have no more than 100 derivation rules, and no more than 100 symbols are used in the derivation rules.

A user must give *max-length* as an integer between `1` and `64`. Given grammar description and maximum length, `langdump` prints each member string to the standard output if its length is no more than given *max-length*. There are no specific constraints on the ordering of the member strings in the printing. `langdump` must return an error message and immediately terminate if the given input is invalid.

Your program must have one or more recursive functions to derive member strings from given derivation rules. You must elaborate the design of the recursive functions in your report. For the other aspects, there no specific constraints.

[1] Such kind of languages is known as *context-free language*.

[2] The grammar description file of the example can be found at http://github.com/hongshin/DiscreteMath/tree/master/assignments/gram.txt

4. Problems

For each of the following problems, find a grammar for constructing a language, and then use your implementation of `langdump` to explore some of its member strings.

(1) Give a grammar of all binary palindrome strings (e.g., `1`, `101`, `0110`), and then generate the member strings up to length 6.

(2) Give a grammar of all strings where one or more parentheses, one or more curly brackets and/or one or more square brackets are well-balanced and properly nested (e.g., `()`, `[]`, `[{()}{}]`). Note that a parenthesis cannot surround a curly or square bracket, and a curly bracket cannot a square bracket. Generate the member strings up to length 10.

(3) Give a grammar of all postfix arithmetic expressions where a number is either `1`, `2`, `3` or `4`, and an operator is either '`+`', '`-`', '`*`' or '`/`'. Generate the member strings up to length 10.

You must give the answer and how `langdump` finds the member strings in your report. You cannot dump the result of `langdump` to the report if there are too many strings. For such cases, you must find another convincing way to show how you checked that `langdump` that works correctly.

5. Submission

Your report must be written in a given template and must not exceed 3 pages. The homework result will be evaluated primary based on your report; thus, you should present all results in detail with clear presentation. In evaluation, the followings are expected to be found from your reports:

- an overview of the program design, and the design of recursive functions for generating member strings
- the representation of derivation rules
- the grammar for each recursive language problem, and the way to confirm that `langdump` generate correct results.

Your implementation of `langdump` must be in one or multiple C source code files. The submitted code must be compatible with GCC 5.4.0 or higher versions. You must give a script to build an executable from the source code.

Upload a Zip file containing all results to **Hisnet** by **11:59 PM, 17 November (Wed)**. The Zip file must contain all files of your language generation program (e.g., source code, build script) and a PDF file of your report. Note that your report must be submitted as a PDF file (not a source file such as MS word file) to avoid formatting errors and other compatibility issues. The submission deadline is strict; no late submission will be accepted after the deadline.