Project 2 Report CFG to CNF Converter in JAVA

Atakan Bodur S018406

Introduction

This document defines the design architecture of the “CFG to CNF Converter” software. Software takes a txt file as CFG output and prints the resulting CNF to the console.

Structures

CFG is designed as a static class and each variable and productions belongs to it is represented by String’s. To achieve this, there is a variable of type Map called that holds variable as the key, and an ArrayList holding all of the productions corresponding to it. The rulesMap, defined above, is initialized during the file reading phase.

CFG also holds all of the inputs stated in the input.txt file and stores them in either Lists or Strings.

Conversion

As you know, conversion has certain rules;

1. Add new start variable if start variable exists on RHS
2. Remove null productions
3. Remove unit productions
4. Handle the RHS

Rule 1:

It’s handled by addNewStartVar() method; it checks for all the RHS exists on the rulesMap and if its true; it creates a newStartVar and initializes it as the start variable of the CFG.

Rule 2:

First, a List is recorded on the method removalOfNullProductions() in order to save which states are directly/indirectly leads to “e”, which is the convention followed by the example inputs.

After determining which variables are nullable, for each nullable var in the list; we find the indexes of the nullable variables in the ruleMap by calculation of combination. This process will initialize the indexOfNullable variable which is type of Map<String, List<List<int>>>, which corresponds to <nullableVariable, <on which production it occurs < in which indexes it occurs>>.

After the indexes of nullables are determined, it creates new productions with traversing over each production the variable in the map points, and removes the indexes where the nullable occurs.The map may supply one or more indexes at once.

Rule 3

Removal of unit productions are done by iterating over each variable on the CFG and determining unit productions it points to, if any, and appends the production of the unit variable to itself.

This process is repeated until no unit productions remain.

Rule 4

RHS modification involves two parts:

1. Separating productions that hold terminals and non-terminals at the same time.
2. Separating productions that’s length is more than two.

First part is done by an algorithm that determines whether the production holds two of different type variables in itself using two for loops and a Boolean. If the Boolean checks out to be “false”, the algorithm separates the string, creates a new variable with production of the new string and adds it to the rulesMap while removing the separated string from the rules list. This process is repeated until the Boolean returns “true”.

Second part is done with the same logic, only difference being the key on the while loop is “production.length()>2”.