CME3203 Theory of Computation

Converting CFGs to Chomsky Normal Form

GROUP

2019510031 ABDULLAH DİNÇ

2019510068 FATİH SEMİRGİN

LECTURER

Assistant Professor ÖZLEM AKTAŞ
Dr. Meltem YILDIRIM EKİCİ

Date 27.12.2022

- **Firstly we need 4 main functions:**

1 - *eliminate_epsilons*();
2 - *eliminate_units*();
3 - *eliminate_terminals*();
4 - *break_long*();

- **But there are more functions which helps us to solve our sub problems, these are ;**

1 - *read_file*("CFG.txt"); // reads the file

2 - *nullables*(*data*); // for determine the nullable variables

3 - add_non_nullable() // addind nullabes into arrlist

4 - check_start() //checking right sides include S

5 - check_nullable_long(String str) //check for variables that only includes nullable variables

6 - check_variable(String arr) // check for any production include any variable

7 - new_data(String temp) //returns new productions after elimination

8 - combination_new_data(String temp) //filling arraylist with all possible productions at combinations class

9 - include_non_nullable(String temp) // to weed out all the possibilities that come

10 - count_non_nullables(String temp, String possibilty) // it works with 9.func, for to weed out all the possibilities that come

11 - check_valid(String temp) // check the eliminating

12 - eliminate_duplicates() // eliminate duplicate production in case they are exist

13 - check_units() //return true if any productions includes unit products

14 - new_variable() // creating new variable to be used

15 - check_terminals(String arr) // determines new productions without terminal

16 - check_long(String str) // return true if productions length greater than 2, then it breaks them

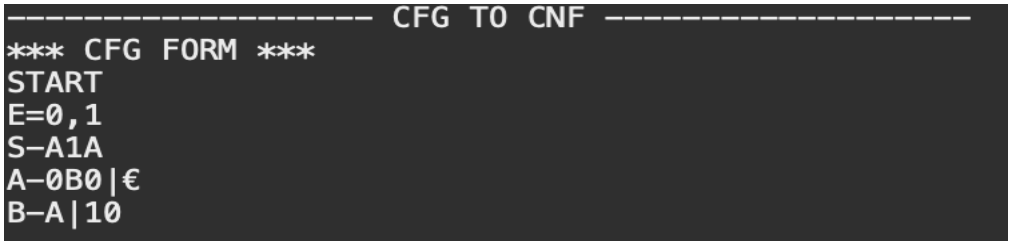
17 - general_check_break() // for to be use 16.func in recursive manner, if production exist that has length greater than 2

18 - print() // for print

- In solution, we used arraylists mostly. We have arraylists which keeps data(what we read from file), alphabet, variables, productions, nullabes and some arraylist which used for temporary things. Our solution based on 5 steps;

1 – Read the txt and create necessary elements for solution.

Here we read our txt line by line, and add to data arraylist without any change.

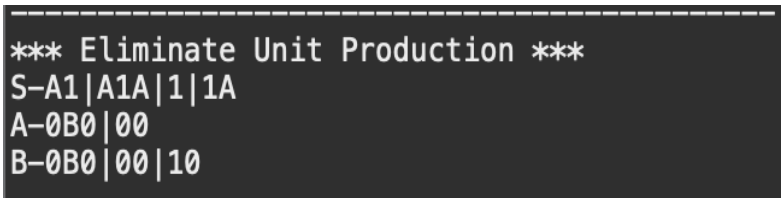


2- Find the €'s and eleminate them.

First we find the symbols which contains epsilon. And we save them into **nullabes.temp** array. If there is a nullable variable contains in symbols, we also add them. Then if the nullable variable is not singular (ex: AB is nullable ,if A and B are nullable) in other variables. Later, we check the if right side includes S positions. Finally we add non-nullables into **non_nullabes** arraylist. At last we eliminate duplicates.

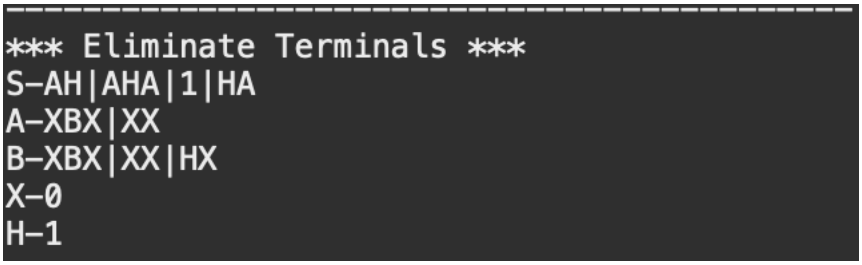
3-Eliminate Units:

Firstly, we delete if the variable contains itself like it's production. Example: A -> AB | A
Then, it becomes to A -> AB . Secondly we delete other unit productions and update productions. If there is a unit productions in any variable, then we call that function again. At last we eliminate duplicates.



4- Eliminate Terminals :

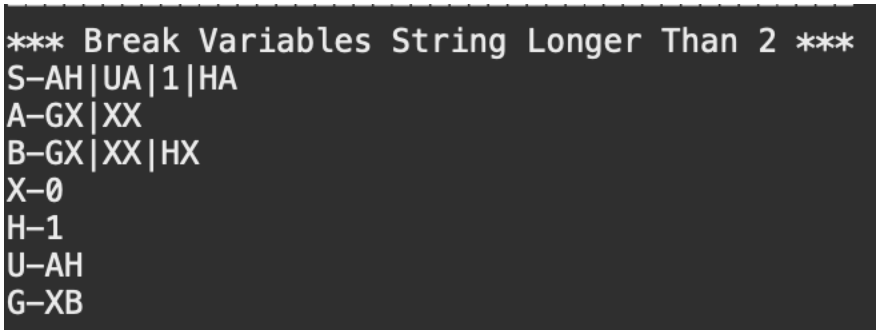
Firstly we add productions which given us as alphabet elements. Thus our terminals became to variables. We find productions which contains terminal element. So they are alphabet. Then we replace terminals that include a production. In final step, we update these productions.



5- Break Productions which has length greater than 2:

We find all productions which has length greater than 2. Then, we creates new productions which consists of first two char's of this long product. Also this new productions has new variables. If this productions already exist for a variable, we do not create new variable because we have it already. If the production is still

have length that greater than 2, it runs again. Lastly, we can eliminate duplicates for any mistakes. It is not obligation but we wanted to check everything.



PSEUDOCODE:

1. Eliminate Epsilons:

1. Call the function `nullables` and store the result in a list called `nullables`
2. Initialize a string called `new_element` to an empty string
3. Initialize a string called `variable` to an empty string
4. If the size of `nullables_temp` is greater than 0, do the following:
   - Iterate over `productions` and do the following:
      - Split the current element using a space as a delimiter and store the result in a list called `split_data`
      - Set `variable` to the element in `variables` at the same index as the current element in `productions`
      - Iterate over `split_data` and do the following:
         - If the current element is a variable (determined by calling `check_variable`) and has a length greater than 1, do the following:
            - Call `new_data` on the current element and store the result in a variable called `new_data_result`
            - If `new_data_result` is equal to an empty string (ignore case), add the current element to `new_element` followed by a space
            - Otherwise, add `new_data_result` to `new_element` followed by a space
         - Otherwise, add the current element to `new_element` followed by a space
      - If `nullables_temp` contains `variable` and `variable` is equal to "S" (ignore case), add "€" to `new_element`
      - Set the element in `productions` at the same index as the current element to `new_element.trim()`
      - Set `new_element` to an empty string
5. Call the function `eliminate_duplicates`
6. Return `productions`

2.Eleminate Unit Productions :

1. Initialize a string called `new_element` to an empty string
2. Initialize a boolean called `flag` to `false`
3. Initialize a string called `variable` to an empty string
4. Initialize an integer called `index` to `-1`
5. Iterate over `productions` and do the following:
   - Split the current element using a space as a delimiter and store the result in a list called `split_data`
   - Set `variable` to the element in `variables` at the same index as the current element in `productions`
   - Iterate over `split_data` and do the following:
      - If the current element is not equal to `variable`, add the current element to `new_element` followed by a space
   - Set the element in `productions` at the same index as the current element to `new_element.trim()`
6. Iterate over `productions` and do the following:
   - Set `new_element` to an empty string
   - Split the current element using a space as a delimiter and store the result in a list called `split_data`
   - Iterate over `split_data` and do the following:
      - If `variables` contains the current element, do the following:
         - Set `index` to the index of the current element in `variables`
         - Add the element in `productions` at index `index` to `new_element` followed by a space
         - Set `flag` to `true`
      - Otherwise, add the current element to `new_element` followed by a space
      - If `flag` is `true` and `j` is equal to the last index of `split_data`, do the following:
         - Set the element in `productions` at the same index as the current element to `new_element.trim()`
7. If the function `check_units` returns `true`, call the function `eliminate_units`
8. Call the function `eliminate_duplicates`
9. Return `productions`

3.Eliminate Terminals:

1. Initialize a string called `new_element` to an empty string
2. Iterate over `alphabet` and do the following:
   - If `productions` does not contain the current element, do the following:
      - Add a new variable to `variables` by calling the function `new_variable`
      - Add the current element to `productions`
3. Initialize a string called `new_element` to an empty string
4. Iterate over `productions` and do the following:
   - Split the current element using a space as a delimiter and store the result in a list called `split_data`
   - Set `new_element` to an empty string

- Iterate over `split_data` and do the following:
    - If the current element has a length greater than 1, do the following:
        - Call the function `check_terminals` on the current element and store the result in a variable called `check_terminals_result`
        - If `check_terminals_result` is not an empty string (ignore case), add `check_terminals_result` to `new_element` followed by a space
        - Otherwise, add the current element to `new_element` followed by a space
    - Otherwise, add the current element to `new_element` followed by a space
  - Set the element in `productions` at the same index as the current element to `new_element.trim()`
5. Return `productions`

4.Break Length greater than 2 :

1. Initialize a string called `new_element` to an empty string
2. Iterate over `productions` and do the following:
  - Split the current element using a space as a delimiter and store the result in a list called `split_data`
  - Set `new_element` to an empty string
  - Iterate over `split_data` and do the following:
    - If the current element has a length greater than 2, do the following:
      - Call the function `check_long` on the current element and store the result in a variable called `check_long_result`
      - Add `check_long_result` to `new_element` followed by a space
    - Otherwise, add the current element to `new_element` followed by a space
  - Set the element in `productions` at the same index as the current element to `new_element.trim()`
3. If the function `general_check_break` returns `true`, call the function `break_long`
4. Call the function `eliminate_duplicates`
5. Return `productions`

Additional Class (Combinations) used for find all possible productions of a product:

START

Step 1
Declare and initialize variables:
- output: a StringBuilder object
- inputstring: a String variable
- possibilities: an ArrayList of String objects

Step 2
Define a constructor that takes a String parameter and assigns it to the inputstring variable

Step 3
Define a method called "combine" that calls the private "combine" method with an input parameter of 0

Step 4
Define the private "combine" method, which takes an integer parameter called "start"

Step 5
For each index "i" in the inputstring, starting at the value of "start":
- Append the character at index "i" in the inputstring to the output StringBuilder object
- Add the current value of the output StringBuilder object to the ArrayList of possibilities
- If "i" is less than the length of the inputstring, call the "combine" method recursively with an input parameter of "i + 1"
- Set the length of the output StringBuilder object to be one character shorter than its current length

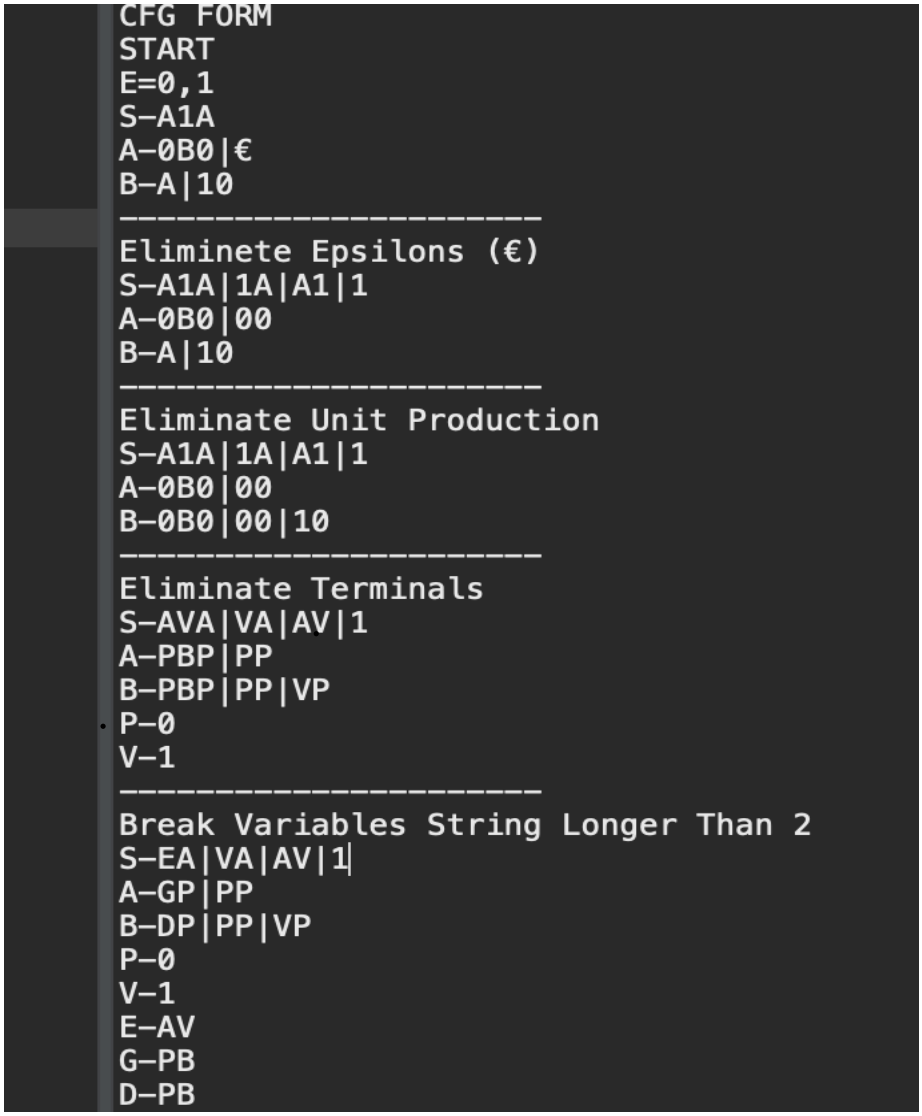
Step 6
Define a method called "clear" that clears the ArrayList of possibilities

Step 7
Define a method called "getInputstring" that returns the value of the inputstring variable

Step 8
Define a method called "setInputstring" that takes a String parameter and assigns it to the inputstring variable

END

THE EXAMPLE:



Reference :

1 - https://javahungry.blogspot.com/2014/02/algorithm-for-combinations-of-string-java-code-with-example.html