ST0270 Formal Languages and Compilers
Assignment 2

Sergio Ramírez Rico

March 10, 2025

1 Deadline

See course website.

2 Assignment

The assignment is to implement, in any programming language, the algorithms to eliminate left-recursion presented in Aho et al. 2006, Section 4.3.3.

You can assume

- the capital letter S is the initial symbol of the grammar,
- nonterminals are represented by uppercase letters and terminals by lowercase letters,
- the symbol `e` represents the empty string (ε), this will be needed for the output,
- the grammar has no cycles, i.e., for no nonterminal A, $A \xrightarrow{+} A$, and
- the grammar ha no ε-productions, i.e., for no nonterminal A, $A \to \varepsilon$.

2.1 Input/Output

Your program should fulfill the following specifications.

Input

A *case* is a context-free grammar $G = (N, \Sigma, P, S)$. The input of the program is as follows.

- A line with a number $n > 0$ indicating how many cases you will receive.
- For each case, a natural number $k > 0$ representing the number of nonterminals ($k = |N|$).
- Then, your program should read k lines with the productions given in the following format:

```
<nonterminal> -> <derivation alternatives of the nonterminal
                  separated by blank spaces>
```

Output

For each case, print an equivalent grammar without left recursion. To print the productions, follow the same format as in the input.

Use single capital letters for the nonterminals to be created, avoid names like `A'`, `S1`, etc. Print a line break between cases.

3 Assignment Submission

Solutions should be submitted using GitHub Classroom. You should follow the instructions below for this assignment.

1. It is allowed to work in groups of no more than two students.
2. Submissions only accepted via the GitHub Classroom link available on course website.
3. Follow the input/output instructions. Do not print extra lines.
4. A `README.md` file (Markdown format) in English is required. It must contain the following information:
   - Full names.
   - Class number.
   - Versions of the operating system, programming language, and tools used in your implementation.
   - Detailed instructions for running your implementation.
   - Explanation of the algorithm.
5. Do not include unnecessary files or directories in the repository.
6. Details about your reference sources apart from the guide book (books, articles, videos, AIs, repositories, etc.).
7. Make sure your code is both clean and organized. For example, it should not contain unnecessary comments or unused code; it should be modularized.

References

Aho, Alfred V. et al. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321486811.

Example I/O

The following are just a few cases where your algorithm should work correctly. Your solution should be strong enough to process any grammar that follows the conditions of the input.

Input

```
3
1
S -> Sa b
2
S -> Aa b
A -> Ac Sd m
2
S -> Sa Ab
A -> Ac Sc c
```

Output

```
S -> bZ
Z -> aZ e

S -> Aa b
A -> bdZ mZ
Z -> cZ adZ e

S -> AbZ
A -> cY
Z -> aZ e
Y -> cY bZcY e
```

Note that there are many equivalent ways to print a grammar.