Program documentation

Author: Aleksander Świniarski

Subject: Program reading grammar (terminals, nonterminals, productions) and removing left recursion from this grammar. Ordering of terminals can be given on input

I. General overview and assumptions

The aim of this project is to create a program which will be able to read CFG grammars provided by the user and remove the left recursion from it if it has one and fulfils conditions for algorithm to run properly.

Assumptions:

- Grammars provided in the input file are context free grammars (CFG)
- Grammars provided in the input file will be written according to the specific format
- Grammar may contain errors which will be handled by the program accordingly
- Grammar does not have to contain left recursion or be proper candidate for algorithm as it will be analysed under this aspects
- Grammar's terminal and nonterminals may be words of various lengths. They cannot be the same as the grammar tokens (e.g.:#G) or comment symbol (//).
- Program will treat "e" by default as an epsilon symbol "ε" but user may change its definition
- Program for the removal of left recursion algorithm will use by default the order of nonterminals that corresponds to the order in which they were defined in the file but user may define its own sequence

II. Functional requirements

- Program is designed as console application
- Program is written using C++ programming language
- Program will read the input grammar from the text file in which the grammar will be written using format defined in the section III.4.a
- Program checks validity of the grammar by checking if:
    - It has defined terminals, nonterminals or productions
    - words are only defined as terminal or nonterminal
    - grammar is CFG
    - productions consists of defined terminals and nonterminals
    - all of nonterminals have their defined productions
    - word for epsilon is uniquely defined
    - Checking if the sequence of nonterminals consists of only and all defined nonterminals
- When the program detects an error in the grammar, it will display the according error message and correct it (e.g.: removing undefined terminal from production) so the

program can continue processing the grammar. In case of critical error, the processing of the grammar will be stopped.

- Program will analyse if the read grammar needs the removal of left recursion. If such situation happens, the program will ensure that grammar fulfils conditions to proceed with the algorithm of left recursion removal.
- Program will produce the file called: modified.txt which will consist of input by user grammars without the left recursion and any detected error. Format of the output file is specified in the section IV.b
- Program can be run in descriptive mode by adding "-d" or "--descriptive" flag to the input. In this mode program will describe for each grammar what definitions have been detected and actions it took (e.g.: "New terminal detected: …", "Grammar after epsilon productions removal: …")

III. Implementation

1. General architecture:

The program can be divided into four main parts which will be done for each grammar included in the input file. Figure III.I.1 presents the workflow of the program and below are short descriptions of each of its segments:

- Reading grammar:

  At first the program will read all of the definitions included for the grammar. Using the set of tokens program can easily identify which symbol represents terminal, nonterminal or productions.

- Error checking of grammar:

  The second step of the program thoroughly checks if the grammar is properly defined by the user. Depending on the type or error, program will resolve it accordingly and inform the user with warning about the error and action taken to resolve it. In the case of critical error the processing of the grammar will be aborted and user will be informed.

- Removal of left recursion:

  Afterwards the grammar is confirmed to be properly defined, the program assess if the grammar contains any left recursion. If so program checks if grammar is viable for removal by checking if it has cycles or ε-productions. If not then the program proceeds with algorithm for removing the left recursion, on the other hand if grammar has cycles or ε-production, program will eliminate them and then return back to check for cycles if any has been produced due to removal of epsilon productions

- Writing grammar into the output file:

  After assuring that the grammar does not have left recursion, the program will write it to the output file: modified.txt

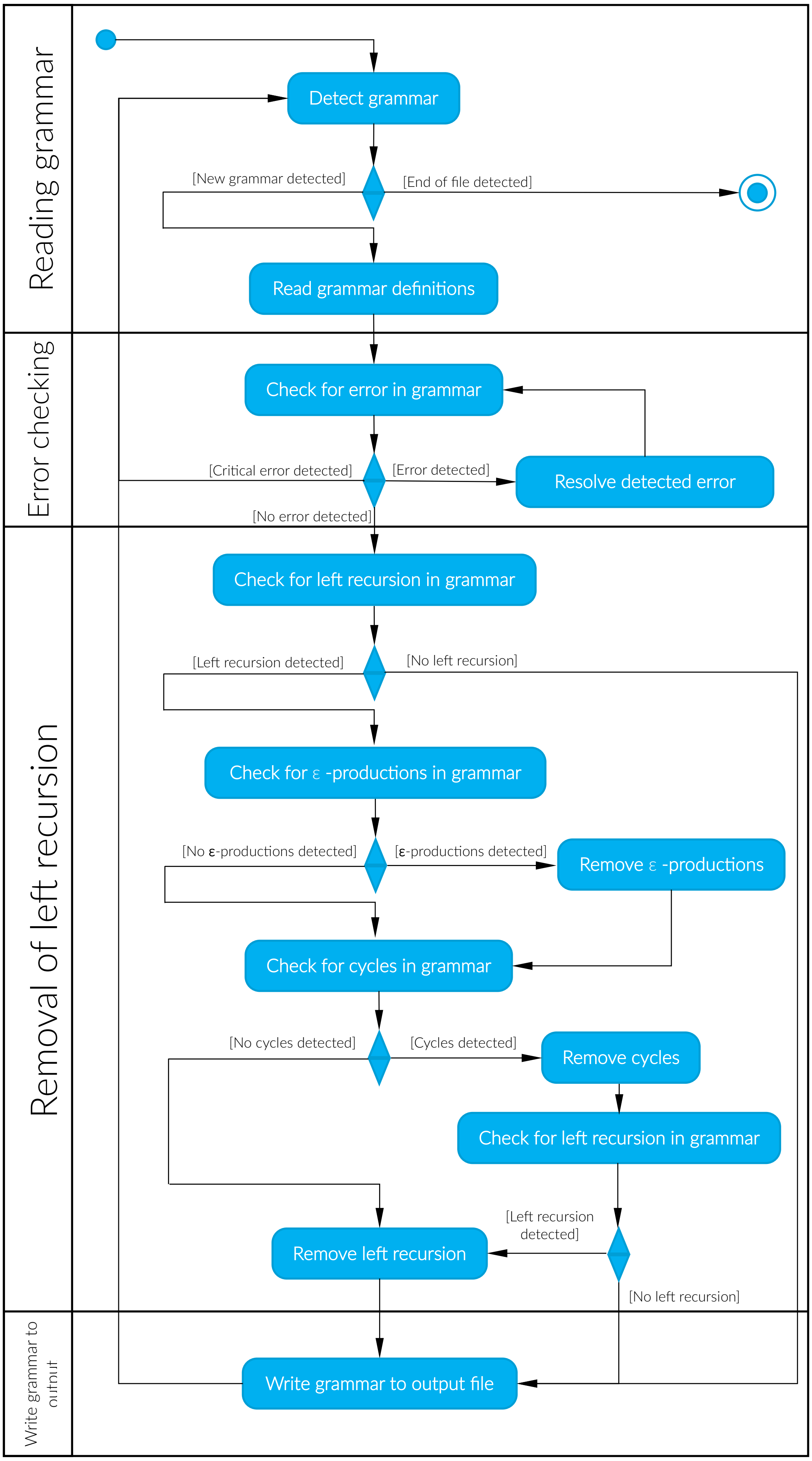

Fig III.1.1 – Activity diagram presenting general architecture of a program

2. Data structures

Grammar – The main object containing data structures related to the definition of the currently processed grammar. Allows for simple and intuitive modifications of the grammar with its methods.

Grammar.Terminals – Set of terminals of the grammar

Grammar.Nonterminals – Set of nonterminals of the grammar

Grammar.Productions – Associative array where each of the productions is associated with a nonterminal

Grammar.Sequence – Vector with sequence of nonterminals used in removal of left recursion

Grammar.Epsilon – Epsilon symbol of a grammar

Grammar.grammarToken – Enum, representing grammar tokens as identifiers

Grammar.MappedGrammarTokens – Associative array which maps grammar tokens in form of strings to enum **grammarToken**.

3. Module descriptions:

All of the main functions/methods needed for the program are included in the Grammar class:

- Grammar.readGrammar() – method which will read the content of the file from the nth instance of the "#G" token till the next its next appearance or end of the file. Method will read word by word. Method by identifying the special tokens like: "#T", "#NT", "#P" knows that the following words should be added to the corresponding data structure. When the method detects comment token ("//") contents till the end of the line will be disregarded

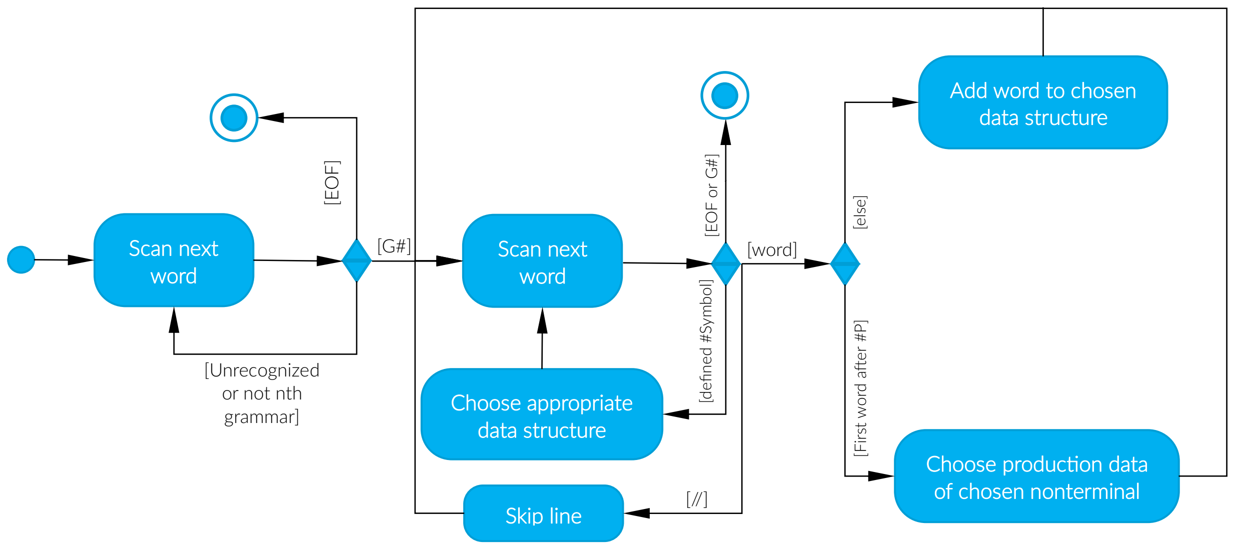
Scan next word
[EOF]
[G#]
[Unrecognized or not nth grammar]
Scan next word
Choose appropriate data structure
Skip line
[EOF or G#]
[defined #Symbol]
[word]
[//]
[else]
[First word after #P]
Add word to chosen data structure
Choose production data of chosen nonterminal

Fig III.3.1 – Activity diagram presenting general workflow of a readGrammar() method

- Grammar.verifyGrammar() – method which purpose is to verify if the scanned from the input grammar is proper CFG grammar. Method will verify following possible errors and solve them accordingly:
  1. Error: Grammar does not have any terminal, nonterminal or productions defined
     - Solution: Critical error which stops processing current grammar and warns user about it
  2. Error: Word defined as a terminal and nonterminal
     - Solution: Method will keep only the terminal definition of the word and inform the user with the warning
  3. Error: Production consists of undefined terminals or nonterminals
     - Solution: Production will be rejected and the user will be informed with the warning
  4. Error: Repetition of productions for the same nonterminals
     - Solution: Method will remove the duplicates and inform the user with the warning
  5. Error: Left side of production is not single defined nonterminal
     - Solution: Method at first will try to detect if the left side production consists of nonterminals. If so the first one from the nonterminals definition will be the final left side of the production. Otherwise the whole production definition will be disregarded
  6. Error: Nonterminal does not have defined any productions
     - Solution: Nonterminal will be disregarded and removed from the rest of the productions. Finally user will be informed with the warning
  7. Error: Epsilon is defined like one of terminals or nonterminals
     - Solution: Epsilon will be reset to the default value or if default value is already defined as terminal or nonterminal, method will look for string of two characters which is not being currently used. Afterwards method will inform about new definition of epsilon in the warning. If the new value of epsilon cannot be resolved the grammar processing will be aborted and user will be informed.
  8. Error: Suggested sequence of nonterminals by user does not contain all nonterminals or consists of not defined ones
     - Solution: Sequence suggested by the user will be disregarded and redefine it to the default one which is the order at which nonterminals are stored in data structure. At the end the method will inform user with warning

  Method after serving an error will start analysing the grammar from the start of the list to ensure that action didn't caused additional errors.

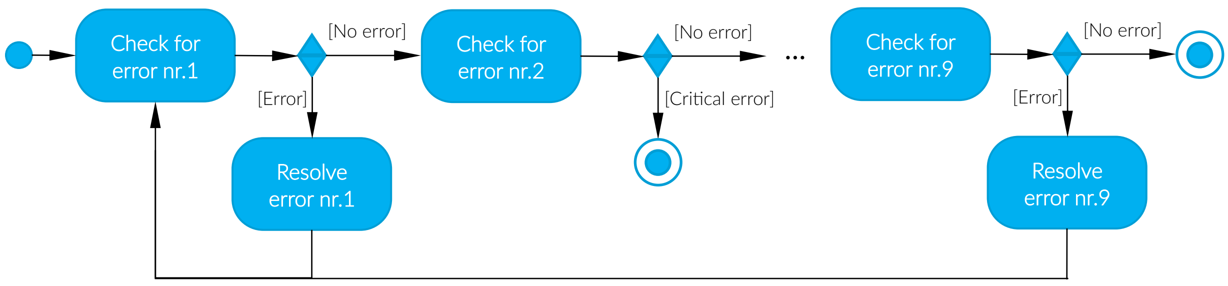

Fig III.3.2 – Activity diagram presenting general workflow of a verifyGrammar() method

- Grammar.checkForLeftRecursion() – Method which will check the productions of the grammar to deduce if it has left recursion. Method will go through all of the productions and if on the right side the first symbol will be corresponding nonterminal, program will proceed with the removal. Otherwise program will skip to writing the grammar to the file.

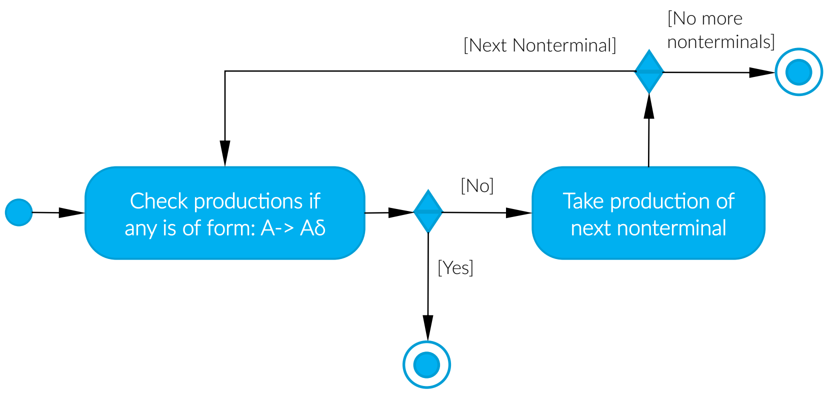
[Next Nonterminal]
[No more nonterminals]
Check productions if any is of form: A-> Aδ
[No]
Take production of next nonterminal
[Yes]

Fig III.3.3 – Activity diagram presenting general workflow of a checkForLeftRecursion() method

- Grammar.handleCycles() – Method will check if the defined grammar consists of Cycles. If so the method will try to remove it from the grammar. Removal of cycles is basically a elimination of unit productions which can be also the simplest form of the left recursion if the verified nonterminal creates only itself in single production. Algorithm will check productions of each nonterminal and eliminate its unit productions which will be replaced by productions of eliminated nonterminal which are not unit productions. If the method will remove the simplest form of left recursion the analysis of the grammar will be finished and program will proceed with writing it to the file.

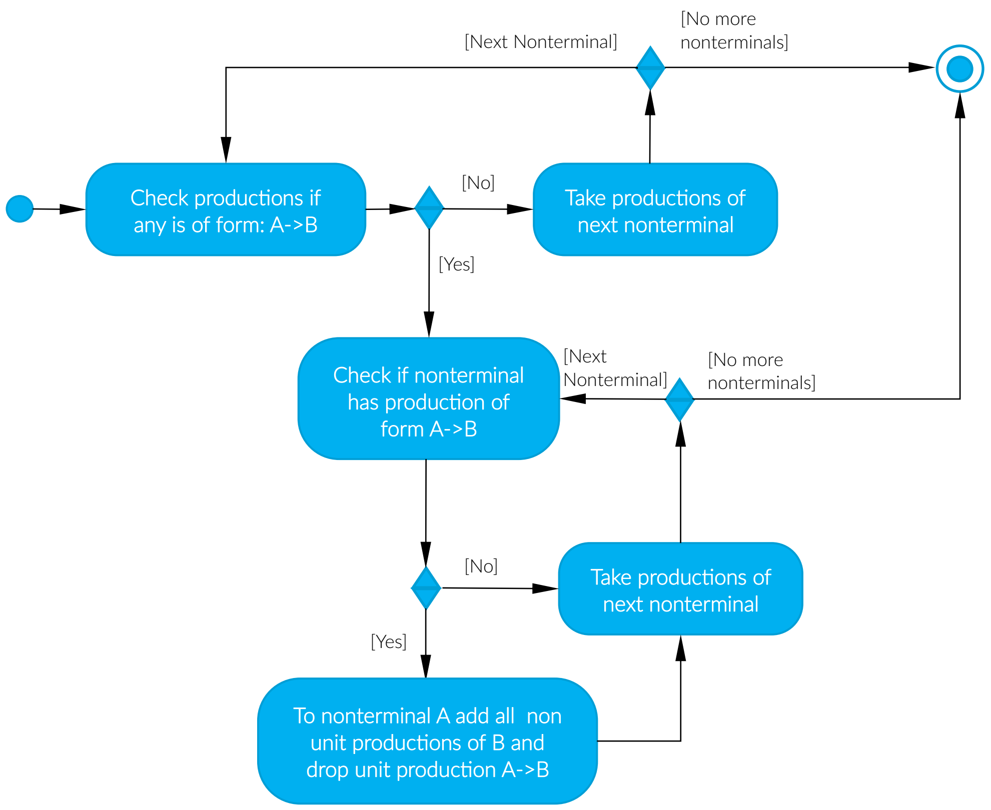
[Next Nonterminal]
[No more nonterminals]
Check productions if any is of form: A->B
[No]
Take productions of next nonterminal
[Yes]
Check if nonterminal has production of form A->B
[Next Nonterminal]
[No more nonterminals]
[No]
Take productions of next nonterminal
[Yes]
To nonterminal A add all non unit productions of B and drop unit production A->B

Fig III.3.3 – Activity diagram presenting general workflow of a handleCycles() method

- Grammar.handleEpsilonProductions() – Method will check if the grammar consists of epsilon productions. If so algorithm will proceed with removing them. At first the algorithm will have to identify nullable nonterminals – it means that nonterminal A have such derivation that: A=>*ε. After identification for each nonterminal we add productions which we can obtain from its already defined productions by removing one or more nullable nonterminals. Finally we remove epsilon productions and obtain equivalent grammar without any epsilon productions.

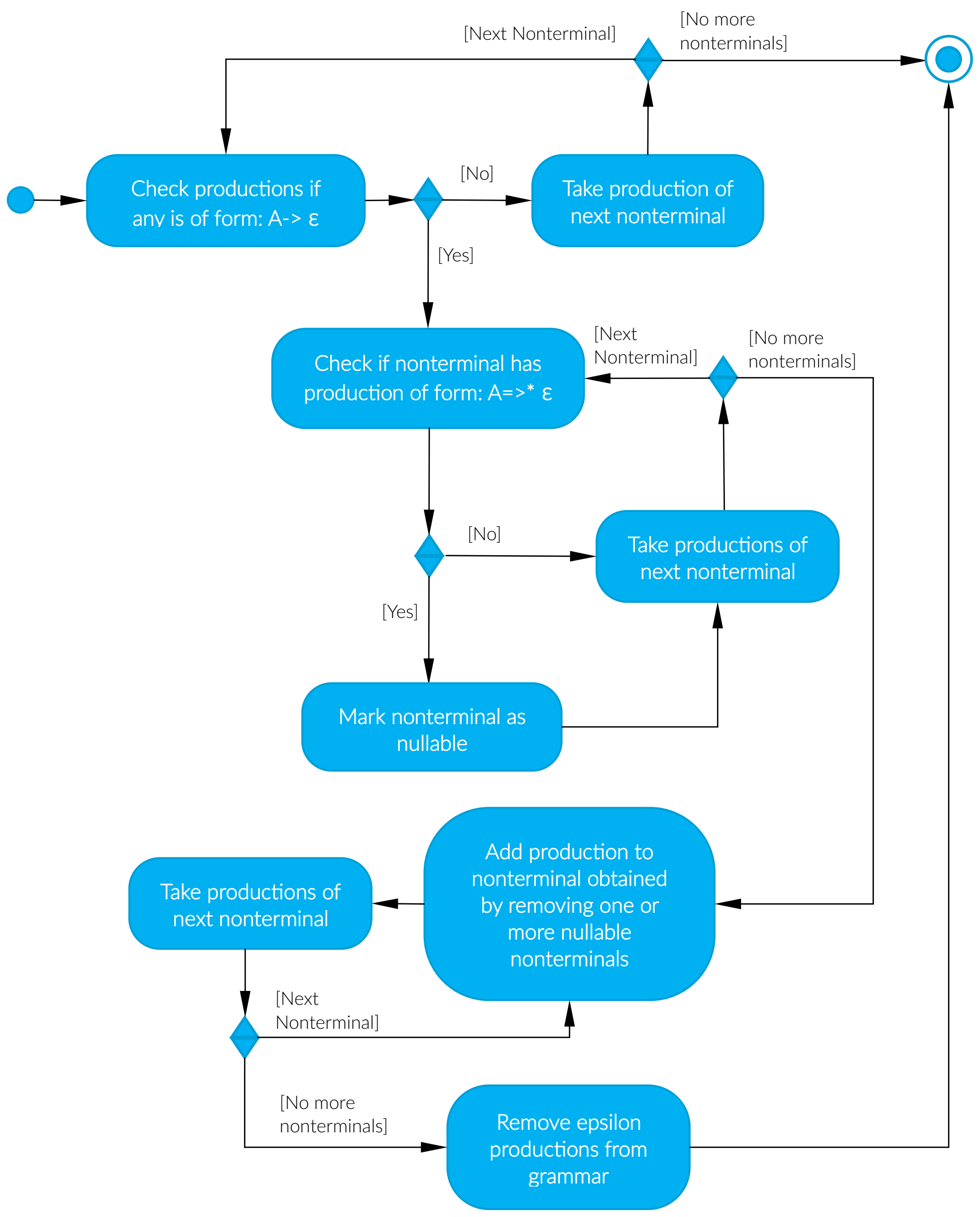
[Next Nonterminal]
[No more nonterminals]
Check productions if any is of form: A-> ε
[No]
Take production of next nonterminal
[Yes]
Check if nonterminal has production of form: A=>* ε
[Next Nonterminal]
[No more nonterminals]
[No]
Take productions of next nonterminal
[Yes]
Mark nonterminal as nullable
Take productions of next nonterminal
Add production to nonterminal obtained by removing one or more nullable nonterminals
[Next Nonterminal]
[No more nonterminals]
Remove epsilon productions from grammar

Fig III.3.3 – Activity diagram presenting general workflow of a handleEpsilonProductions() method

- Grammar.removeLeftRecursion() – To remove left-recursion from the grammar the algorithm will have to follow some order of nonterminals. By default it is in the order nonterminals were defined in the file but the user can define its own. Then for all nonterminals starting from the beginning of the order, each production of the form $A \to A_j\gamma$, where A_j are nonterminals located before the nonterminal A in the order, we replace with production $A \to \delta_1\gamma \mid \delta_2\gamma \mid \dots \mid \delta_k\gamma$ where $A_j \to \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$. In the next step algorithm will remove immediate recursion among productions of nonterminal A by grouping them as: $A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$ and replace them by: $A \to \beta_1A' \mid \beta_2A' \mid \dots \mid \beta_nA'$ and $A' \to \alpha_1A' \mid \alpha_2A' \mid \dots \mid \alpha_mA' \mid \varepsilon$.

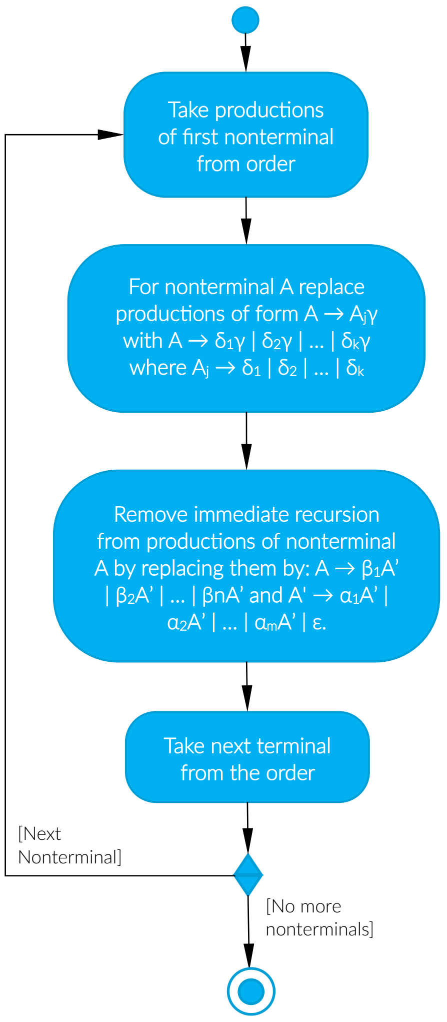
Take productions of first nonterminal from order
For nonterminal A replace productions of form $A \to A_j\gamma$ with $A \to \delta_1\gamma \mid \delta_2\gamma \mid \dots \mid \delta_k\gamma$ where $A_j \to \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$
Remove immediate recursion from productions of nonterminal A by replacing them by: $A \to \beta_1A' \mid \beta_2A' \mid \dots \mid \beta nA'$ and $A' \to \alpha_1A' \mid \alpha_2A' \mid \dots \mid \alpha_mA' \mid \varepsilon$.
Take next terminal from the order
[Next Nonterminal]
[No more nonterminals]

Fig III.3.3 – Activity diagram presenting general workflow of a removeLeftRecursion() method

- Grammar.writeGrammarToFile() – Method which will write the processed grammar to the output file in the specified format defined in the section III.4.b
- Grammar.erase() – Simple method which will be called before reading new grammar from the file to clear all of the data structures

4. Input/output description:

a. Input

Program expects maximally two types of parameters at the input:

- Descriptive flag: [-d|--descriptive] – User by specifying the flag in the input turns on the mode which will make program describe its actions. The detailed output of program with this flag set is specified in the section III.IV.b.
- Text file with grammars formatted using special tokens:
    - // - Token indicates beginning of comment which makes program ignore the content of the file till end of the line
    - #G – Token indicates start of definition of new grammar. All of the following tokens will be treated as the part of this grammar till the new definition of grammar or end of the file.
    - #Eps – (Optional) Indicates that the following word will be new definition of epsilon. It can be defined multiple times but only the last instance will be used as a definition of the epsilon. Only one non-token word will be taken into consideration after #Eps token as a new definition of epsilon.
    - #NT – Token indicates beginning of definitions of nonterminals of a grammar
    - #T – Token indicates beginning of definitions of terminals of a grammar
    - #P – Token indicates beginning of productions of chosen nonterminal. First word is nonterminal for which productions will be defined. Each of the productions has to be divided with "|" and each word in production with spacebar " "
    - #Seq – (Optional) Token indicates ordering of terminals that the user wants to use in the removal of left recursion from the grammar. If multiple sequences will be defined, only the last one will be used
    - Tokens can be used multiple times and can be interleaved
    - Tokens that are stated before start of grammar definition will be disregarded
    - Tokens does not have to be stated at the beginning of the line. Each line can consist of many definitions
    - Each word and token has to be separated using spacebar

```
// Definition of the first grammar
#G                         // Beginning of new grammar
#Eps eps                   // Optional new definition of epsilon
#NT A B C                  // Definition of non terminals of grammar
#T a b c                   // Definition of terminals of grammar
#P A a | b | c | eps       // Definition of 4 productions of nonterminal A
#P B a A b | b C c         // Definition of 2 production of nonterminal B
#P C C a b c               // Definition of production of nonterminal C
#Seq C B A                 // Definition of nonterminal sequence for algorithm

// Definition of the second grammar
#G
#T answer babble call #NT FIRST SECOND THIRD
#P FIRST answer babble
#P THIRD call | e
#P SECOND babble FIRST call
#NT dance                  // Interleaved definitions
#T FOURTH                  // of terminals and nonterminals
#P FOURTH answer | dance | e
```

Fig. III.4.a.1: Exemplary .txt file with properly defined grammars

b. Output

Program will output two following sets of data:

- Console output – Program by default will inform user about its intermediate steps and warnings concerning errors about grammar specified in part III.3. Additionally during the descriptive mode which can be turned on at the input, program will inform about all of its actions and results of them. Below are described all of output strings that program can produce and situations when they will be produced:
  - Information about intermediate steps
    In the following output strings the $X represents the number of grammar to which the info relates to
    - `Scanning grammar nr.[$X]` – Program starts scanning definitions of the diagram
    - `Error checking grammar nr.[$X]` – Output sent at the start of error checking of the grammar
    - `Handling epsilon production from grammar nr.[$X]` – Output sent when epsilon productions are being detected in the grammar and if so they are being removed
    - `Handling cycles from grammar nr.[$X]` – Program detects cycles in the grammar and if they exists, proceeds to remove them
    - `Removing left recursion from grammar nr.[$X]` – Output sent whenever the program will try to remove present in grammar left recursion and grammar fulfils requirements of the algorithm
    - `Writing grammar nr.[$X] to the file` – Output string sent when program writes left recursion-free grammar to the output file
  - Warnings
    All of the following output messages will be preceded by the "`{Error}`" or "`{CRITICAL ERROR}`" string depending on malignancy of error. Behaviour of program regarding following errors is specified in section III.3
    - `Grammar does not have any: [terminal | nonterminal | productions] defined! It is being rejected` – Critical warning informing that due to the lack of definition of terminals, nonterminals or productions, current grammar will not be handled
    - `[Word] has definition of terminal and nonterminal. [Word] will be removed from the nonterminals definition of the grammar` – Warning informing about double definition of the word in the grammar
    - `[Production] consists of undefined word: [undefined word]. Production will be removed` – Warning informing of usage of undefined terminal or nonterminal in the production
    - `[production] repeated. Duplicate will be removed` – Warning informing about removal of duplicates of production

        - `Left side of production: [left side of production] is not a single defined nonterminal. [Within was detected nonterminal: [nonterminal]. [Productions are moved to detected nonterminal | Productions for [nonterminal] nonterminal are being removed]` – Warning informing about action taken in the case of improperly defined left side of the production
        - `[nonterminal] does not have defined productions. Nonterminal will be removed` – Waning informing about the lack of production for one of the grammars' nonterminals
        - `[epsilon definition] is also defined as terminal or nonterminal. [Epsilon has been restored to the [default | generated] value | Unable to restore epsilon value. Terminated processing of a grammar!]` – Warning informing that the word used to define epsilon is also used to define terminal or nonterminal
        - `Sequence [sequence of nonterminals] is improperly defined. Using default sequence of nonterminals` – Warning informing about using not defined nonterminals in the sequence for algorithm
    - Action descriptions output strings (only in descriptive mode)
        - `New terminal detected: [list of terminals]` – Sent whenever program will detect new terminal definitions for currently processed grammar
        - `New nonterminal detected: [list of nonterminals]` – Sent whenever program will detect new nonterminal definitions for currently processed grammar
        - `New production for [Nonterminal] detected: [list of productions]` – Sent whenever program will detect new production definitions for nonterminal of currently processed grammar
        - `New definition of epsilon: [new epsilon definition]` – Sent when program detects new definition of epsilon token for currently processed grammar
        - `New order of nonterminals detected: [order of nonterminals]` – Sent when program detects order of nonterminals by user to be used in the algorithm of removal of left recursion
        - `Grammar after [error | cycles | epsilon production | left recursion] removal: [Grammar after modification]` – Output which will present grammar after modification which happened due to actions of the program. Grammar will be presented according to the format in which it will be written in the file (next subpoint) but without the line specifying its number

- The text file called: modified.txt file consisting of grammars without detected errors and left recursion. Grammars are written according to the following format where n is number of grammars a k is number of nonterminals in the grammar:

```
Grammar 1:
Nonterminals: [All of nonterminals divided with spacebar]
Terminals: [All of terminals divided with spacebar]
Productions:
       [Nonterminal 1] -> [Productions of nonterminal divided with |]
       [Nonterminal 2] -> [Productions of nonterminal divided with |]
       …
       [Nonterminal k]  ->[Productions of nonterminal divided with |]

Grammar n+1:
Nonterminals: [All of nonterminals divided with spacebar]
Terminals: [All of terminals divided with spacebar]
Productions:
       [Nonterminal 1] -> [Productions of nonterminal divided with |]
       [Nonterminal 2] -> [Productions of nonterminal divided with |]
       …
       [Nonterminal k] -> [Productions of nonterminal divided with |]
```

Fig. III.4.b.1: Format of writing grammars into the output file

```
Grammar 1:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
       1. S -> a | /\ | ( T )
       2. T -> S T’ | S
       3. T’ -> , S T’ | ε

Grammar 2:
Nonterminals: A B S A’ S’
Terminals: a + * ( )
Productions:
       1. S -> A S' | A
       2. S' -> + A | + A S'
       3. A -> B A' | B
       4. A' -> * B | * B A
       5. B -> ( S ) | a
```

Fig. III.4.b.2: Exemplary modify.txt presenting possible output file

IV. Functional test cases

1. File without any grammar
   Input: Text file without any grammar definitions

```
//Empty file
```

Fig. IV.1.1: input.txt containing no characters for first test case

   Expected: modify.txt will not be created

2. File with properly defined grammars
   Input: Text file with 5 properly defined grammars:

    1. with left recursion, no cycles and epsilon productions
    2. without left recursion
    3. with left recursion and cycles and no epsilon productions
    4. with left recursion and epsilon productions and no cycles
    5. with left recursion, cycles and epsilon productions

```
// Grammar 1

#G
#T a /\ ( ) ,
#NT S T
#P S a | /\ | ( T )
#P T S | T , S

// Grammar 2

#G
#T a b
#NT A B
#P A a B a | b A b
#P B b B b | a A a

// Grammar 3

#G
#T a b
#NT S A
#P S a A a | S a | A
#P A b | S

// Grammar 4

#G
#T a b
#NT S A B
#P S A S B
#P A A a S | a
#P B a | b

// Grammar 5

#G
#T a b
#NT S A B
#P S A S B | e
#P A a A S | a | B
#P B B b S | A | B b
```

Fig. IV.2.1: input.txt containing 5 grammars for second test case

Expected:

- Modify.txt file with consisting of 5 grammars without left recursions, cycles and original epsilon productions.

```
Grammar 1:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 2:
Nonterminals: A B
Terminals: a b
Productions:
      1. A -> a B a | b A b
      2. B -> b B b | a A a

Grammar 3:
Nonterminals: S A S'
Terminals: a b
Productions:
      1. A -> b | a A a | a A a S' a | b S' a
      2. S -> a A a S' | b S'
      3. S' -> a S' | e

Grammar 4:
Nonterminals: S A B A'
Terminals: a b
Productions:
      1. A -> a A'
      2. A' -> a S A' | e
      3. B -> a | b
      4. S -> A S B

Grammar 5:
Nonterminals: S A B B'
Terminals: a b
Productions:
      1. A -> a A | a A S | a | B b | B b S
      2. B -> a A B' | a A S B' | a B'
      3. B' -> b B' | b S B' | b B' | e
      4. S -> A B | A S B
```
Fig. IV.2.2: modified.txt containing 5 processed grammars

- Output in the console informing about intermediate steps taken for each grammar:

```
Scanning grammar nr.1
Error checking grammar nr.1
Handling epsilon productions from grammar nr.1
Handling cycles from grammar nr.1
Removing left recursion from grammar nr.1
Writing grammar nr.1 to the file
Scanning grammar nr.2
Error checking grammar nr.2
Writing grammar nr.2 to the file
Scanning grammar nr.3
Error checking grammar nr.3
Handling epsilon productions from grammar nr.3
Handling cycles from grammar nr.3
Removing left recursion from grammar nr.3
Writing grammar nr.3 to the file
Scanning grammar nr.4
Error checking grammar nr.4
Handling epsilon productions from grammar nr.4
Handling cycles from grammar nr.4
Removing left recursion from grammar nr.4
Writing grammar nr.4 to the file
Scanning grammar nr.5
Error checking grammar nr.5
Handling epsilon productions from grammar nr.5
Handling cycles from grammar nr.5
Removing left recursion from grammar nr.5
Writing grammar nr.5 to the file
```
Fig. IV.2.3: Console output informing about each intermediate step for each grammar

3. File with properly defined grammar run in descriptive mode
   Input: Text file with properly defined grammar consisting of cycles, epsilon productions and left recursion

```
#G
#T a b
#NT S A B
#Eps eps
#P S A S B | e
#P A A a S | a | B
#P B B b S | A | B b
#Seq S A B
```
Fig. IV.3.1: input.txt containing grammar with cycles and epsilon production and left recursion for third case

Expected:

- Modify.txt with grammar without left recursion, cycles and original epsilon productions

```
Grammar 1:
Nonterminals: S A B B'
Terminals: a b
Productions:
      1. A -> a A | a A S | a | B b | B b S | b
      2. B -> b B' | a A B' | a A S B' | a B'
      3. B' -> b B' | b S B' | e
      4. S -> A B | A S B
```

Fig. IV.3.2: modified.txt containing processed grammar run in descriptive mode

- Output in the console informing about all of actions taken for a grammar:
    - Scanning grammar
    - Terminal definitions
    - Nonterminal definitions
    - Productions for each nonterminal
    - New definition of epsilon
    - New order of terminals
    - Error checking
    - Removing cycles
    - Removing epsilon productions
    - Removing left recursion
    - Writing grammar

```
Scanning grammar nr.1
        New terminal detected: a
        New terminal detected: b
        New nonterminal detected: S
        New nonterminal detected: A
        New nonterminal detected: B
        New definition of epsilon: eps
        New production for S detected: A S B
        New production for S detected: e
        New production for A detected: a A S
        New production for A detected: a
        New production for A detected: B
        New production for B detected: B b S
        New production for B detected: A
        New production for B detected: B
        New production for B detected: b
        New order of nonterminals detected: S A B
Error checking grammar nr.1
Grammar after verification:
Nonterminals: S A B
Terminals: a b
Productions:
        A -> a A S | a | B
        B -> B b S | A | B | b
        S -> A S B | e

Handling epsilon productions from grammar nr.1
Grammar after handling of epsilon productions:
Nonterminals: S A B
Terminals: a b
Productions:
        A -> a A | a A S | a | B
        B -> B b | B b S | A | B | b
        S -> A B | A S B

Handling cycles from grammar nr.1
Grammar after handling cycles:
Nonterminals: S A B
Terminals: a b
Productions:
        A -> a A | a A S | a | B b | B b S | b
        B -> B b | B b S | b | a A | a A S | a
        S -> A B | A S B
```

```
Removing left recursion from grammar nr.1
Grammar after left recursion removal:
Nonterminals: S A B B'
Terminals: a b
Productions:
        A -> a A | a A S | a | B b | B b S | b
        B -> b B' | a A B' | a A S B' | a B'
        B' -> b B' | b S B' | eps
        S -> A B | A S B

Writing grammar nr.1 to the file
```

Fig. IV.3.3 – Console output of program run in the descriptive mode

4. File with improperly defined grammars

Input: Text file with improperly defined grammars:

1. without terminals
2. without nonterminals
3. without productions
4. with word defined as terminal and nonterminal
5. with production with undefined terminals or nonterminals
6. with word defined as epsilon and terminal or nonterminal
7. with sequence of nonterminals consisting of not defined nonterminals
8. with repeated productions for the same nonterminal
9. in which left side of production is not defined nonterminal
10. in which one nonterminal has no defined productions

```
// Grammar 1

#G
#NT S T
#P S a | /\ | ( T )
#P T S | T , S

// Grammar 2

#G
#T a /\ ( ) ,
#P S a | /\ | ( T )
#P T S | T , S

// Grammar 3

#G
#T a /\ ( ) ,
#NT S T

// Grammar 4

#G
#T a /\ ( ) , S
#NT S T
#P S a | /\ | ( T )
#P T S | T , S

// Grammar 5

#G
#T a /\ ( ) ,
#NT S T
#P S a | /\ | ( T ) | ( undefined )
#P T S | T , S

// Grammar 6

#G
#Eps newEps
#T a /\ ( ) , newEps
#NT S T
#P S a | /\ | ( T )
```

```
#P T S | T , S

// Grammar 7

#G
#T a /\ ( ) ,
#NT S T
#P S a | /\ | ( T )
#P T S | T , S
#Seq A S T

// Grammar 8

#G
#T a /\ ( ) ,
#NT S T
#P S a | /\ | ( T ) | a | /\
#P T S | T , S
#P S /\

// Grammar 9

#G
#T a /\ ( ) ,
#NT S T
#P ST a | /\ | ( T )
#P aT S | T , S

// Grammar 10

#G
#T a /\ ( ) ,
#NT S T A
#P S a | /\ | ( T )
#P T S | T , S
```

Fig. IV.4.1: input.txt containing improperly defined grammars for fourth test case

Expected:

- modify.txt file with grammars from witch it was possible to remove errors and left recursion

```
Grammar 4:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 5:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 6:
Nonterminals: S T T'
Terminals: a /\ ( ) , newEps
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 7:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e
```

```
Grammar 8:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 9:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e

Grammar 10:
Nonterminals: S T T'
Terminals: a /\ ( ) ,
Productions:
      1. S -> a | /\ | ( T )
      2. T -> a T' | /\ T' | ( T ) T'
      3. T' -> , S T' | e
```

Fig. IV.4.2 – modified.txt containing errorless grammars for which critical error wasn't identified

- Console output warnings for each grammar about:
  - Critical error of lack of terminals and not processing grammar
  - Critical error of lack of nonterminals and not processing grammar
  - Critical error of lack of productions and not processing grammar
  - Word defined as nonterminal and terminal
  - Production with undefined words
  - Word being defined as epsilon and terminal or nonterminal
  - Sequence of nonterminals consisting of undefined ones
  - Duplicates of productions for nonterminal
  - Improperly defined production where left side is not a single nonterminal
  - Nonterminal not having any productions and being disregarded

```
Scanning grammar nr.1
Error checking grammar nr.1
        {CRITICAL ERROR} Grammar does not have any terminals defined! It is being
rejected!
Scanning grammar nr.2
Error checking grammar nr.2
        {CRITICAL ERROR} Grammar does not have any non terminals defined! It is
being rejected!
Scanning grammar nr.3
Error checking grammar nr.3
        {CRITICAL ERROR} Grammar does not have any productions defined! It is
being rejected!
Scanning grammar nr.4
Error checking grammar nr.4
        {Error} /\ has definition of terminal and nonterminal. /\ will be removed
from the nonterminals definition of the grammar.
Handling epsilon productions from grammar nr.4
Handling cycles from grammar nr.4
Removing left recursion from grammar nr.4
Writing grammar nr.4 to the file
Scanning grammar nr.5
Error checking grammar nr.5
        {Error} Production: ( undefined ) consists of undefined word: undefined.
Production will be removed
Handling epsilon productions from grammar nr.5
Handling cycles from grammar nr.5
Removing left recursion from grammar nr.5
Writing grammar nr.5 to the file
Scanning grammar nr.6
Error checking grammar nr.6
        {Error} newEps is also defined as terminal or nonterminal. Epsilon has
been restored to the default value!
Handling epsilon productions from grammar nr.6
Handling cycles from grammar nr.6
Removing left recursion from grammar nr.6
Writing grammar nr.6 to the file
Scanning grammar nr.7
Error checking grammar nr.7
        {Error} Sequence: A S T is improperly defined. Using default sequence of
nonterminals
Handling epsilon productions from grammar nr.7
Handling cycles from grammar nr.7
Removing left recursion from grammar nr.7
Writing grammar nr.7 to the file
Scanning grammar nr.8
Error checking grammar nr.8
        {Error} a production repeated. Duplicate will be removed.
        {Error} /\ production repeated. Duplicate will be removed.
        {Error} /\ production repeated. Duplicate will be removed.
Handling epsilon productions from grammar nr.8
Handling cycles from grammar nr.8
Removing left recursion from grammar nr.8
Writing grammar nr.8 to the file
Scanning grammar nr.9
Error checking grammar nr.9
        {Error} Left side of production: O is not a single defined nonterminal.
Productions for O nonterminal are being removed!
        {Error} Left side of production: aT is not a single defined nonterminal.
Within was detected nonterminal: T. Productions are moved to detected nonterminal.
Productions for aT nonterminal are being removed!
        {Error} Left side of production: ST is not a single defined nonterminal.
Within was detected nonterminal: S. Productions are moved to detected nonterminal.
Productions for ST nonterminal are being removed!
Handling epsilon productions from grammar nr.9
Handling cycles from grammar nr.9
Removing left recursion from grammar nr.9
Writing grammar nr.9 to the file
Scanning grammar nr.10
Error checking grammar nr.10
        {Error} A does not have defined productions. Nonterminal will be removed
Handling epsilon productions from grammar nr.10
Handling cycles from grammar nr.10
Removing left recursion from grammar nr.10
Writing grammar nr.10 to the file
```

Fig.IV.4.3 – Console output informing about specific error for specific grammar