**Київський національний університет імені Тараса Шевченка**

Звіт

Лабораторна робота

контекстно - вільні граматики (Prolog)

З дисципліни «Парадигми та технології програмування»

Виконав

студент групи ТТП-3

Каплюк Владислав

Постановка задачі

Варіант 4: Виявити праворекурсивні нетермінали.

Опис алгоритму

Нетермінал A КС-граматики G називається праворекурсивним, якщо в граматиці існує виведення A => wA\*.

В програмі граматика задається як алфавіт нетермінальних символів, алфавіт термінальних символів, правила переходів, аксіома.

Спочатку я шукаю множину епсілон-нетерміналів.

*Означення*: Нетермінал  КС-граматики  називається -нетерміналом, якщо .

Далі для кожного нетермінального символа який стоїть зліва у переході я шукаю всі нетермінальні символи що стоять на останній позиції праворуч у переході, але також я беру не лише останній нетермінал, а я перевіряю чи належить він множині е-нетерміналів у випадку якщо так, то я буду перевіряти попередній чи є він епсілон нетерміналом і додавати в список. Далі я буду прати уже отриманий список і для кожного елемента з нього продовжувати пошук. Якщо на якісь з ітерацій в цьому списку я побачу що додався нетермінал з якого я починав пошук значить існує перехід самого в себе. І Цей нетермінал є праворекурсивний.

Тобто алгоритм розділений на 2 частини

1. Пошук епсілон нетерміналів

*Алгоритм.*Пошук -нетерміналів:





….



…. Пn



Тоді множина  — множина -нетерміналів.

1. Пошук праворекурсивних нетерміналів

*Алгоритм****.*** Тестування нетермінала на праву рекурсію. Для кожного нетермінала Аi побудуємо наступну послідовність множин *S0, S1,* ….

, починаємо з нетерміналу *Аi*.

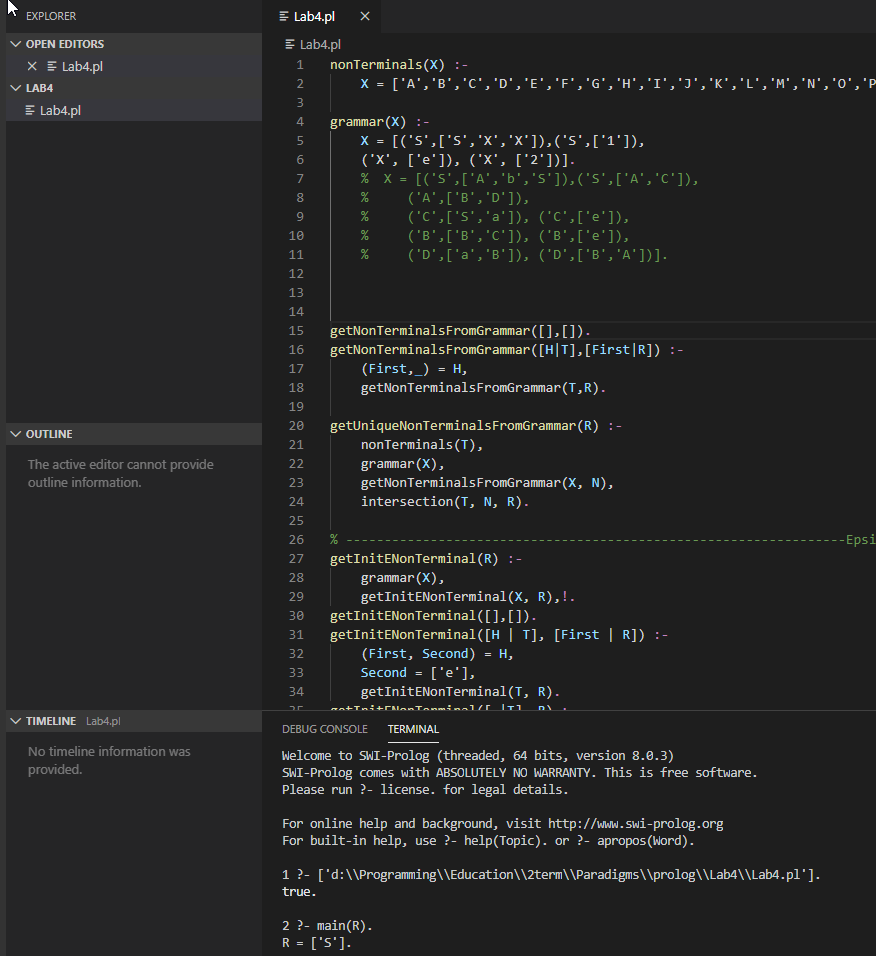
….

Тоді якщо , то  — праворекурсивний нетермінал.

Тести

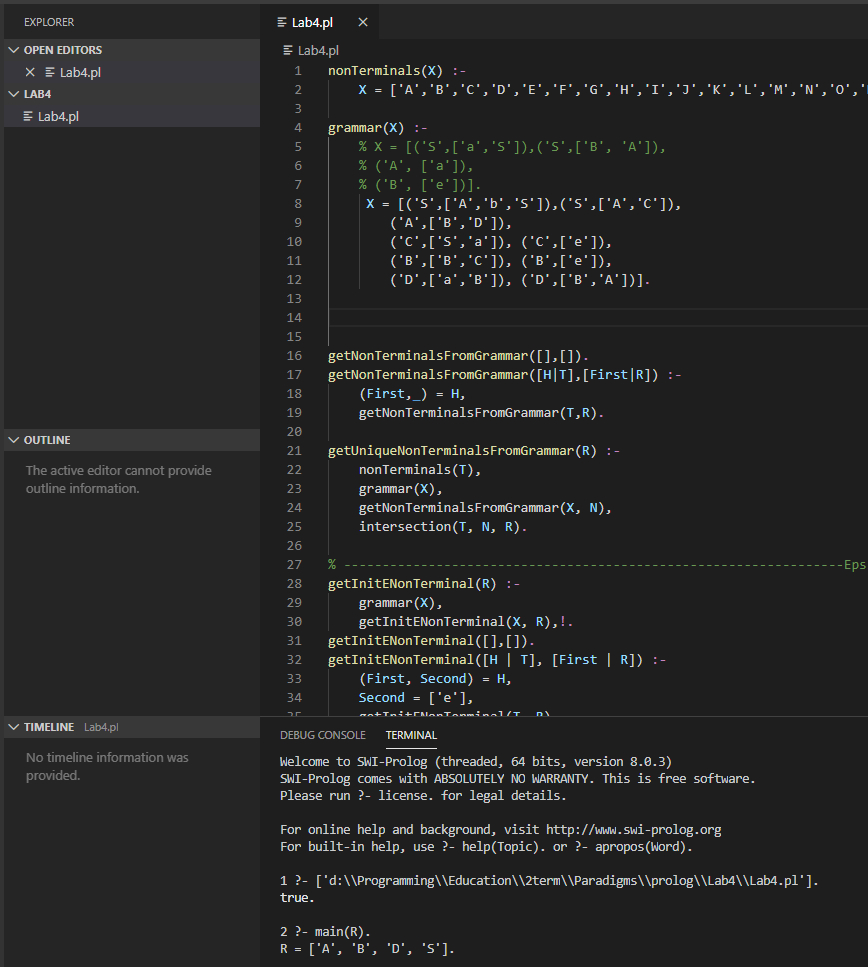
S -> SXX | 1

X -> ε | 2



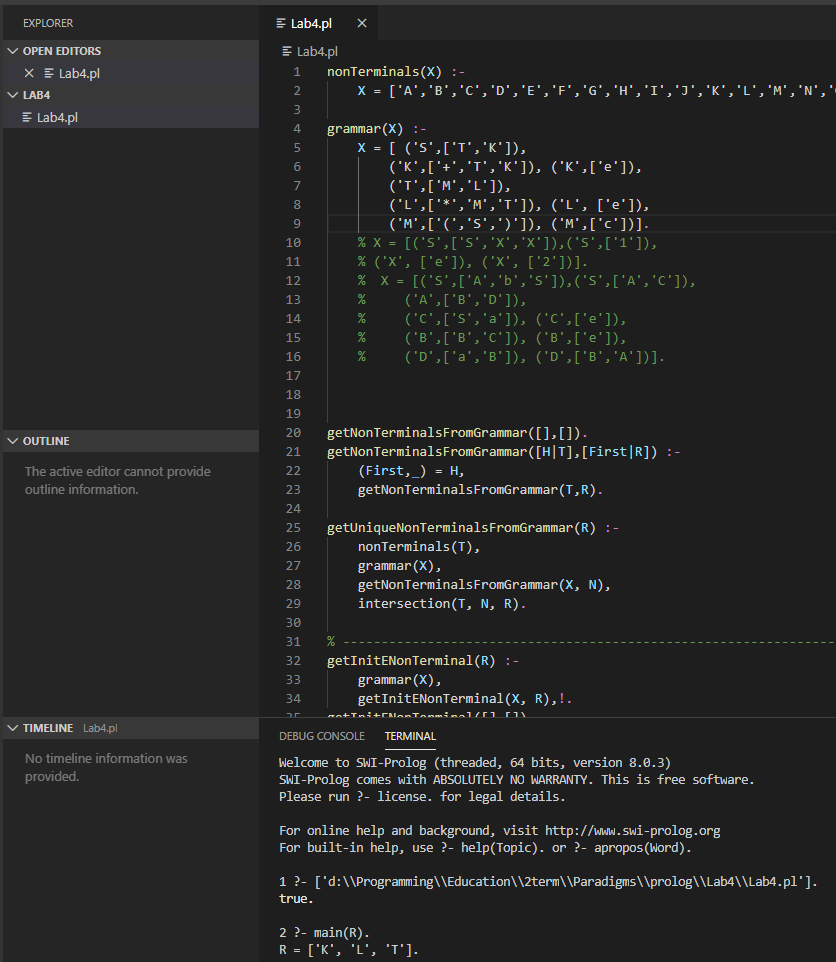
Result : S – праворекурсивний нетермінал





Result: A, B, D, S – праворекурсивні нетермінали

S→TK  
K→+TK ⎪ε (ε – порожнє слово)  
T→ML  
L→\*MT ⎪ε (ε – порожнє слово)  
M→(S) ⎪ c

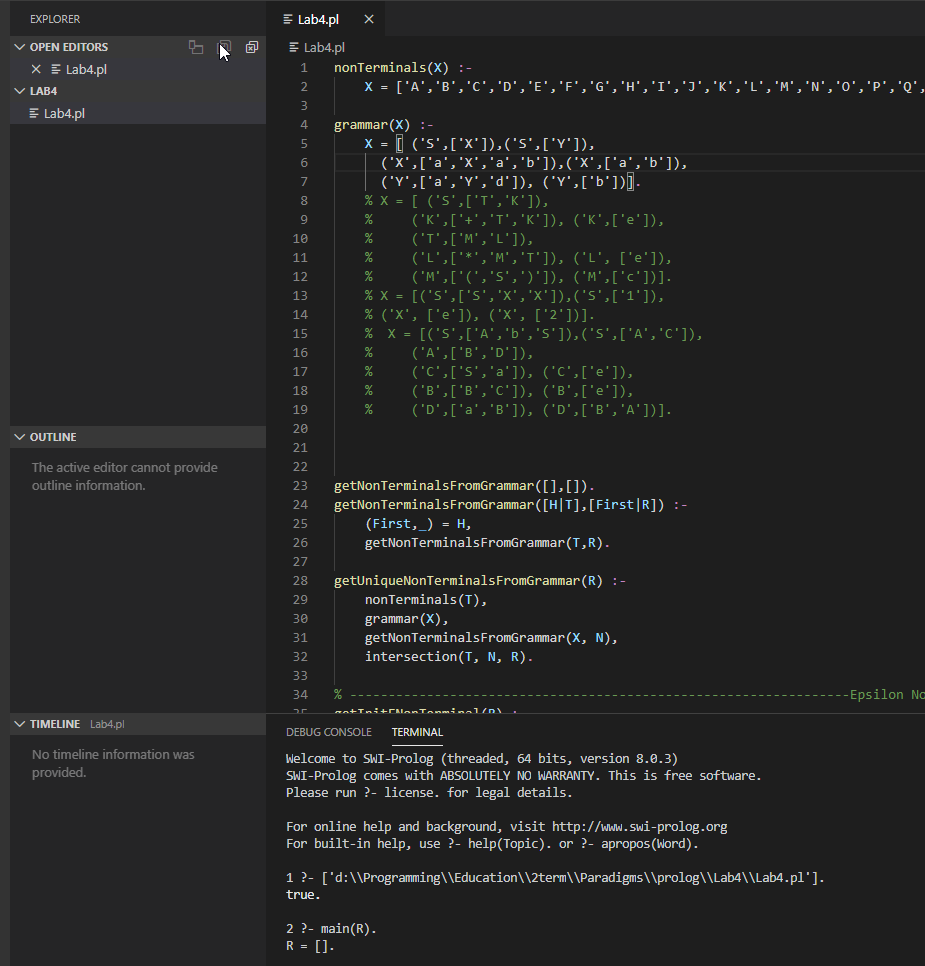


Result: K, L, T – праворекурсивні нетермінали

S→X⎪Y

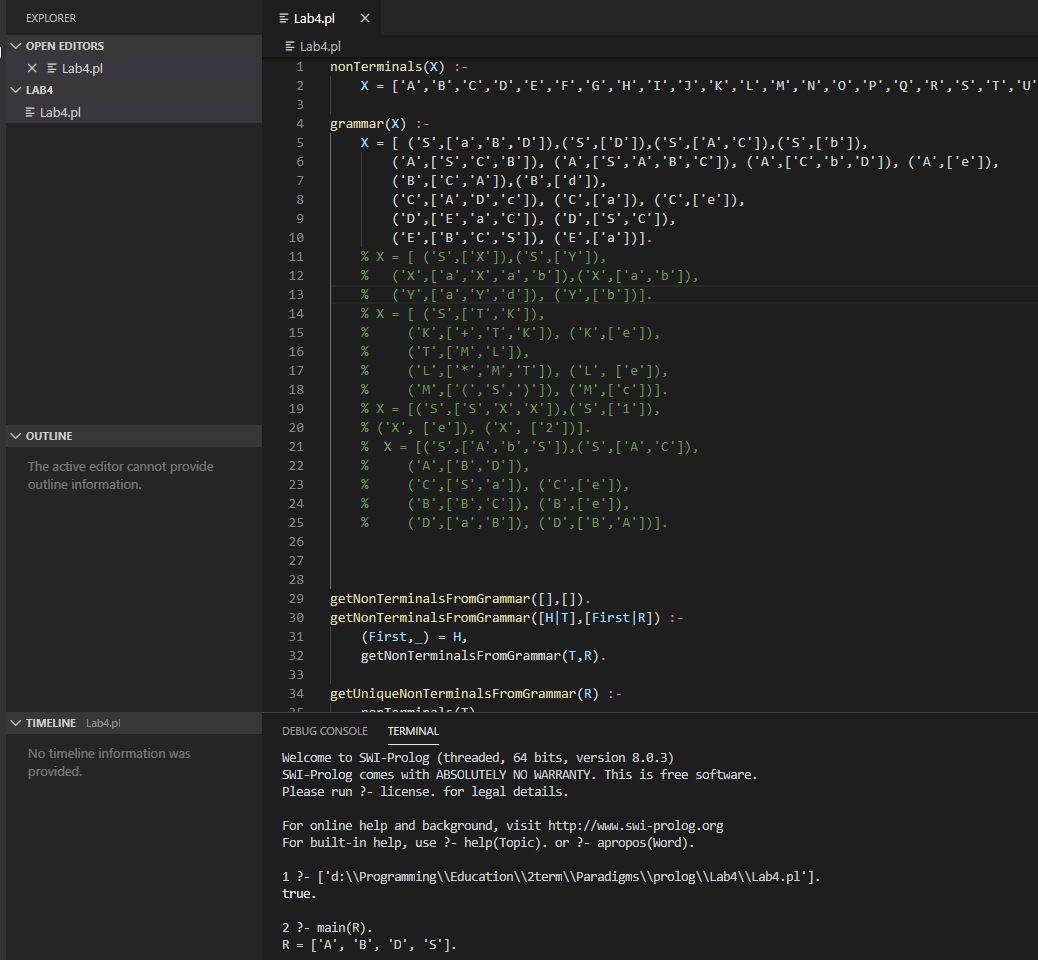
X→aXab⎪ab

Y→aYd⎪b



Result: праворекурсивні нетермінали відсутні





Result: A, B, D, S – праворекурсивні нетермінали

**Код програми**

nonTerminals(X) :-

    X = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'].

grammar(X) :-

    X = [ ('S',['a','B','D']),('S',['D']),('S',['A','C']),('S',['b']),

        ('A',['S','C','B']), ('A',['S','A','B','C']), ('A',['C','b','D']), ('A',['e']),

        ('B',['C','A']),('B',['d']),

        ('C',['A','D','c']), ('C',['a']), ('C',['e']),

        ('D',['E','a','C']), ('D',['S','C']),

        ('E',['B','C','S']), ('E',['a'])].

    % X = [ ('S',['X']),('S',['Y']),

    %   ('X',['a','X','a','b']),('X',['a','b']),

    %   ('Y',['a','Y','d']), ('Y',['b'])].

    % X = [ ('S',['T','K']),

    %     ('K',['+','T','K']), ('K',['e']),

    %     ('T',['M','L']),

    %     ('L',['\*','M','T']), ('L', ['e']),

    %     ('M',['(','S',')']), ('M',['c'])].

    % X = [('S',['S','X','X']),('S',['1']),

    % ('X', ['e']), ('X', ['2'])].

    %  X = [('S',['A','b','S']),('S',['A','C']),

    %     ('A',['B','D']),

    %     ('C',['S','a']), ('C',['e']),

    %     ('B',['B','C']), ('B',['e']),

    %     ('D',['a','B']), ('D',['B','A'])].

getNonTerminalsFromGrammar([],[]).

getNonTerminalsFromGrammar([H|T],[First|R]) :-

    (First,\_) = H,

    getNonTerminalsFromGrammar(T,R).

getUniqueNonTerminalsFromGrammar(R) :-

    nonTerminals(T),

    grammar(X),

    getNonTerminalsFromGrammar(X, N),

    intersection(T, N, R).

% -----------------------------------------------------------------Epsilon Non Terminals------------------------------------

getInitENonTerminal(R) :-

    grammar(X),

    getInitENonTerminal(X, R),!.

getInitENonTerminal([],[]).

getInitENonTerminal([H | T], [First | R]) :-

    (First, Second) = H,

    Second = ['e'],

    getInitENonTerminal(T, R).

getInitENonTerminal([\_|T], R) :-

    getInitENonTerminal(T, R).

getCurrentENonTerminals(ListOfENonTerminals, R) :-

    grammar(X),

    getCurrentENonTerminals(ListOfENonTerminals, X, R).

getCurrentENonTerminals(\_,[],[]).

getCurrentENonTerminals(ListOfENonTerminals, [H | T], [First | R]) :-

    (First, Second) = H,

    intersection(Second, ListOfENonTerminals, Res),

    Res = Second,

    not(member(First, ListOfENonTerminals)),

    getCurrentENonTerminals(ListOfENonTerminals, T, R).

getCurrentENonTerminals(ListOfENonTerminals, [\_ | T], R) :-

    getCurrentENonTerminals(ListOfENonTerminals,T,R).

getEpsilonNonTerminals(ListOfEpsilonNonTerminals, ListOfEpsilonNonTerminals) :-

    getCurrentENonTerminals(ListOfEpsilonNonTerminals, ListOfCurrentENonTerminals),

    length(ListOfCurrentENonTerminals, Int),

    Int is 0.

getEpsilonNonTerminals(ListOfEpsilonNonTerminals, R) :-

    getCurrentENonTerminals(ListOfEpsilonNonTerminals, ListOfCurrentENonTerminals),

    append(ListOfEpsilonNonTerminals, ListOfCurrentENonTerminals, List1AndList2),

    getEpsilonNonTerminals(List1AndList2, R).

%---------------------------------------------------------------------------------------------------------------------------------

getPossibleRules(CurrentNonTerminals, R):-

    grammar(X),

    getPossibleRules(CurrentNonTerminals, X, R),!.

getPossibleRules(\_, [], []).

getPossibleRules(CurrentNonTerminals, [H | T], [H | RT]):-

    (First, Second) = H,

    member(First, CurrentNonTerminals),

    last(Second, Last),

    nonTerminals(X),

    member(Last, X),

    getPossibleRules(CurrentNonTerminals, T, RT),!.

getPossibleRules(CurrentNonTerminals, [\_| T], R):-

    getPossibleRules(CurrentNonTerminals, T, R),!.

getAdjustedRules(\_, \_, -1, []).

getAdjustedRules(Word, ENonTerminals, Current, R):-

    length(Word, Int),

    T is Int - 1, Current is T,

    nth0(Current, Word, Elem),

    member(Elem, ENonTerminals),

    Prev is Current - 1,

    getAdjustedRules(Word, ENonTerminals, Prev, NR),!,

    append([Elem], NR, R).

getAdjustedRules(Word, \_, Current, R):-

    length(Word, Int),

    T is Int - 1, Current is T,

    nth0(Current, Word, Elem),

    append([],[Elem],R).

getAdjustedRules(Word, ENonTerminals, Current, R):-

    Current >= 0,

    nth0(Current, Word, Elem),

    member(Elem, ENonTerminals),

    Prev is Current - 1,

    getAdjustedRules(Word, ENonTerminals, Prev, NR),!,

    append([Elem], NR, R).

getAdjustedRules(Word, \_, Current, R):-

    Current >= 0,

    nth0(Current, Word, Elem),

    nonTerminals(NonTerminals),

    member(Elem, NonTerminals),

    append([],[Elem],R).

getRulesToBeChecked(Rules, ENonTerminals, Result) :-

    getRulesToBeCheckedHelper(Rules, ENonTerminals, R),!,

    append(R, List),

    nonTerminals(NonTerminals),

    intersection(NonTerminals, List, Result),!.

getRulesToBeCheckedHelper([],\_,[]).

getRulesToBeCheckedHelper([H | T], ENonTerminals, [HR | TR]) :-

    (\_, Second) = H,

    length(Second, Length),

    Prev is Length - 1,

    getAdjustedRules(Second, ENonTerminals, Prev, HR),!,

    getRulesToBeCheckedHelper(T, ENonTerminals,TR),!.

getNewRulesToBeChecked([],\_,[]).

getNewRulesToBeChecked([H | T], ExistingRules, [H|RT]) :-

    not(member(H,ExistingRules)),

    getNewRulesToBeChecked(T, ExistingRules, RT),!.

getNewRulesToBeChecked([\_ | T], ExistingRules, R) :-

    getNewRulesToBeChecked(T, ExistingRules, R),!.

getReachableStates(CurrentNonTerminals, ENonTerminals, CurrentNonTerminals) :-

    getPossibleRules(CurrentNonTerminals, PossibleRules),

    getRulesToBeChecked(PossibleRules, ENonTerminals, RulesToBeChecked),

    getNewRulesToBeChecked(RulesToBeChecked, CurrentNonTerminals, NewRulesToBeChecked),

    length(NewRulesToBeChecked, Length),

    Length is 0.

getReachableStates(CurrentNonTerminals, ENonTerminals, R) :-

    getPossibleRules(CurrentNonTerminals, PossibleRules),

    getRulesToBeChecked(PossibleRules, CurrentNonTerminals, RulesToBeChecked),

    getNewRulesToBeChecked(RulesToBeChecked, CurrentNonTerminals, NewRulesToBeChecked),

    append(CurrentNonTerminals, NewRulesToBeChecked, NewCurrentNonTerminals),

    getReachableStates(NewCurrentNonTerminals, ENonTerminals, R).

hasRecursionForCurrentNonTerminal(Current, ENonTerminals) :-

    getPossibleRules([Current], PossibleRules),!,

    getRulesToBeChecked(PossibleRules, ENonTerminals, RulesToBeChecked),!,

    getReachableStates(RulesToBeChecked, ENonTerminals, ReachableStates),!,

    member(Current, ReachableStates).

hasRightRecursiveNonTerminals(NonTerminalsToBeChecked, Current, ENonTerminals, R) :-

    nth0(Current, NonTerminalsToBeChecked, NonTerminal),

    hasRecursionForCurrentNonTerminal(NonTerminal, ENonTerminals),

    length(NonTerminalsToBeChecked, NonTerminalsToBeCheckedLength),

    Next is Current + 1,

    Next < NonTerminalsToBeCheckedLength,

    hasRightRecursiveNonTerminals(NonTerminalsToBeChecked, Next, ENonTerminals, Res),

    append([NonTerminal],Res,R).

hasRightRecursiveNonTerminals(NonTerminalsToBeChecked, Current, ENonTerminals, [NonTerminal]) :-

    nth0(Current, NonTerminalsToBeChecked, NonTerminal),

    hasRecursionForCurrentNonTerminal(NonTerminal, ENonTerminals),

    length(NonTerminalsToBeChecked, NonTerminalsToBeCheckedLength),

    Next is Current + 1,

    Next >= NonTerminalsToBeCheckedLength.

hasRightRecursiveNonTerminals(NonTerminalsToBeChecked, Current, ENonTerminals, R) :-

    length(NonTerminalsToBeChecked, NonTerminalsToBeCheckedLength),

    Next is Current + 1,

    Next < NonTerminalsToBeCheckedLength,

    hasRightRecursiveNonTerminals(NonTerminalsToBeChecked, Next, ENonTerminals, R).

hasRightRecursiveNonTerminals(\_, \_, \_, []).

main(R) :-

    getInitENonTerminal(InitENonTerminals),

    getEpsilonNonTerminals(InitENonTerminals, EpsilonNonTerminals),

    getUniqueNonTerminalsFromGrammar(UniqueNonTerminalsFromGrammar),

    hasRightRecursiveNonTerminals(UniqueNonTerminalsFromGrammar,0, EpsilonNonTerminals,R),!.