|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления ​​​

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии ​​​

**Отчет по лабораторной работе**

**«Распознавание цепочек регулярного языка»**

**по курсу «Конструирование компиляторов»**

**Вариант 5**

Выполнил студент группы ИУ7-21М Доманов К. И.

Проверил Ступников А. А.

*2022 г.*

# Оглавление

Описание задания3

Преобразование регулярного выражения непосредственно в ДКА4

Построение КА с минимальным количеством состояний8

Моделирование КА для входной цепочки из терминалов исходной грамматики10

Текст программы11

Список литературы17

**Описание задания**

Цель работы: приобретение практических навыков реализации важнейших элементов лексических анализаторов на примере распознавания цепочек регулярного языка.

В процессе выполнения лабораторной работы в соответствии с вариантом 5, необходимо написать программу, которая в качестве входа принимает произвольное регулярное выражение и выполняет следующие преобразования:

1. Преобразует регулярное выражение непосредственно в ДКА.
2. По ДКА строит эквивалентный ему КА, имеющий наименьшее возможное количество состояний. Для выполнения данного задания необходимо воспользоваться алгоритмом разбиения состояний на классы эквивалентности, который приведен по ссылке [Минимизация ДКА, алгоритм за O(n^2) с построением пар различимых состояний — Викиконспекты (ifmo.ru)](http://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%94%D0%9A%D0%90,_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B7%D0%B0_O(n%5E2)_%D1%81_%D0%BF%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC_%D0%BF%D0%B0%D1%80_%D1%80%D0%B0%D0%B7%D0%BB%D0%B8%D1%87%D0%B8%D0%BC%D1%8B%D1%85_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B9).
3. Моделирует минимальный КА для входной цепочки из терминалов исходной грамматики.

**Преобразование регулярного выражения непосредственно в ДКА**

Пусть дано регулярное выражение r в алфавите T. К регулярному выражению r добавим маркер конца: (r)#. Такое регулярное выражение будем называть пополненным. В процессе своей работы алгоритм будет использовать пополненное регулярное выражение.

Алгоритм будет оперировать с синтаксическим деревом для пополненного регулярного выражения (r)# , каждый лист которого помечен символом a http://citforum.ru/programming/theory/serebryakov/cmsy10-32.gif T http://citforum.ru/programming/theory/serebryakov/cmsy10-5b.gif{e, #}, а каждая внутренняя вершина помечена знаком одной из операций: **.** (конкатенация), | (объединение), \* (итерация).

Каждому листу дерева (кроме e-листьев) припишем уникальный номер, называемый позицией, и будем использовать его, с одной стороны, для ссылки на лист в дереве, и, с другой стороны, для ссылки на символ, соответствующий этому листу. Заметим, что если некоторый символ используется в регулярном выражении несколько раз, он имеет несколько позиций.

Теперь, обходя дерево T снизу-вверх слева-направо, вычислим четыре функции: nullable, firstpos, lastpos и followpos. Функции nullable, firstpos и lastpos определены на узлах дерева, а followpos - на множестве позиций. Значением всех функций, кроме nullable, является множество позиций. Функция followpos вычисляется через три остальные функции [1].

Таблица для вычисления функций представлена на рисунке 1.

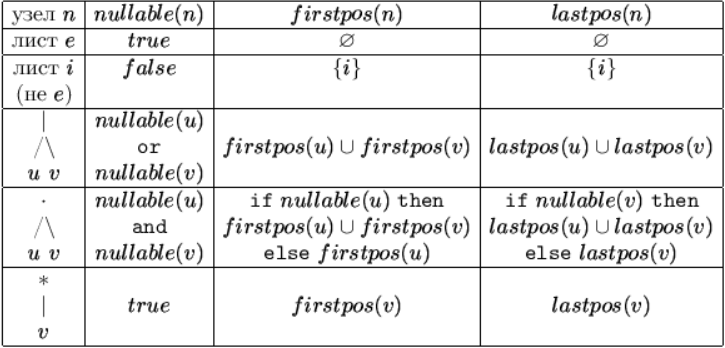


Рисунок 1 – Таблица для вычислений nullable, firspos и lastpos

Синтаксическое дерево для пополненного регулярного выражения (a|b)\*abb# с результатом вычисления функций firstpos и lastpos приведено на рисунке 2. Слева от каждого узла расположено значение firstpos, справа от узла - значение lastpos.

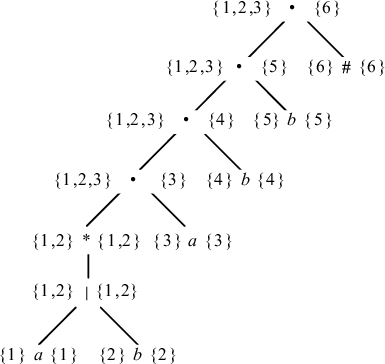


Рисунок 2 – Результат вычисления функций firstpos и lastpos

Для вычисления followpos используются 2 правила:

1. Если n – cat-узел с левым потомком c1 и правым c2, то для каждой из позиций i из lastpos(c1) все позиции из firstpos(c2) содержатся в followpos(i).
2. Если n – star-узел и i-ая позиция в lastpos(n), то все позиции из firstpos(n) содержатся в followpos(i) [2].

Таблица расчета функции followpos представлена на рисунке 3.

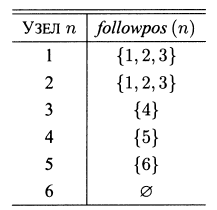


Рисунок 3 – Результат вычисления функции followpos

На следующем этапе необходимо построить множество состояний ДКА и функцию переходов. Изначально все состояния непомечены. Помеченым состояние становится непосредственно перед тем, как мы рассматриваем его исходящие переходы. Начальным состоянием является firstpos(n0), где узел n0 – корень T. Принимающими являются состояния, которые содержат позицию для символа ограничителя #.

ДКА для регулярного выражения r = (a|b)\*abb представлен на рисунке 4.

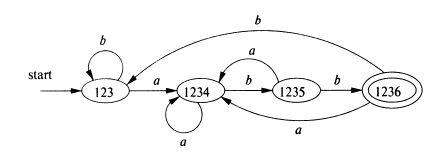


Рисунок 4 – ДКА для регулярного выражения r = (a|b)\*abb

Результат работы программы для регулярного выражения r = (a|b)\*abb представлен на рисунке 5.

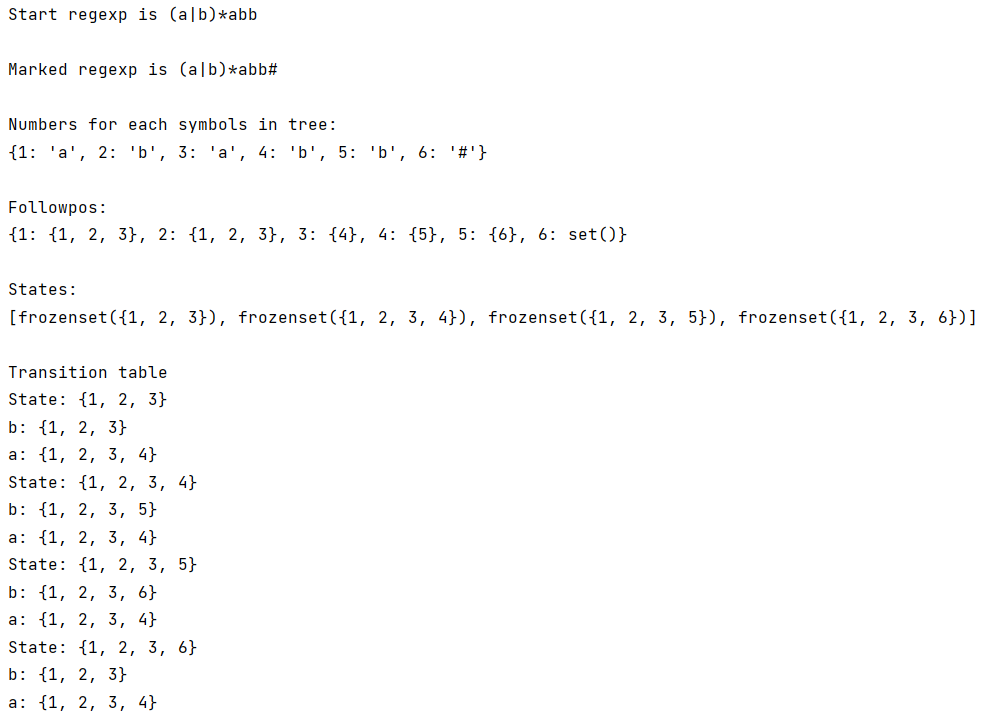


Рисунок 5 – Результат работы программы для r = (a|b)\*abb

Результат работы программы для регулярного выражения r = abcbbd представлен на рисунках 6 и 7.

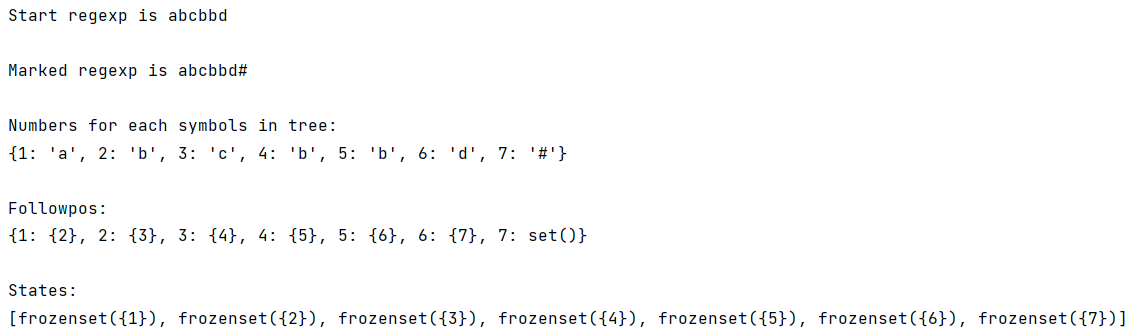


Рисунок 6 – Результат работы программы для r = abcbbd

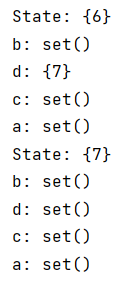
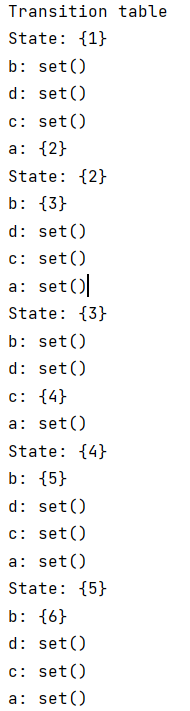


Рисунок 7 – Результат работы программы для r = abcbbd

Таким образом, можно сделать вывод о правильной реализации алгоритма построения ДКА из регулярного выражения.

**Построение КА с минимальным количеством состояний**

В данном пункте необходимо реализовать алгоритм минимизации ДКА с построением пар различных состояний. Если под воздействием любой входной цепочки, автоматы попадают оба либо в пару финальных, либо в пару нефинальных состояний, то данные состояния являются эквивалентными. Пример построения КА с минимальным количеством состояний представлен на рисунке 8.

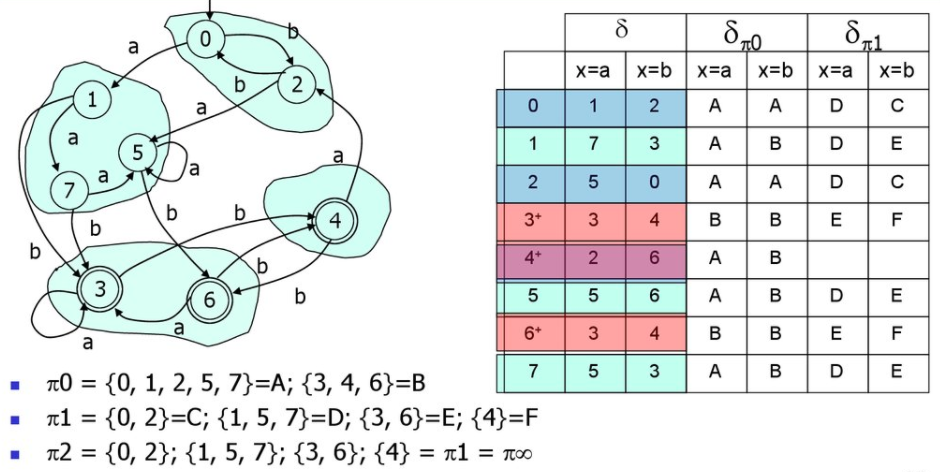


Рисунок 8 – Пример построения КА с минимальным количеством состояний

Манипуляции с отношениями эквивалентности удобно выполнять с помощью матрицы инциденций, в которой в клетке <i, j> ставим N, если элементы i и j не принадлежат одному и тому же блоку соответствующего разбиения. Пример построения матрицы инциденций представлен на рисунке 9.



Рисунок 9 – Пример построения матрицы инциденций

В ходе лабораторной работы, был реализован алгоритм минимизации ДКА. Результат минимизации ДКА для r = (a|b)\*abb представлен на рисунке 10.

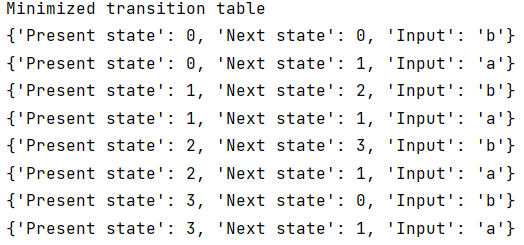


Рисунок 10 – Пример минимизации ДКА для r = (a|b)\*abb

Результат минимизации ДКА для r = abcbdb представлен на рисунке 11.

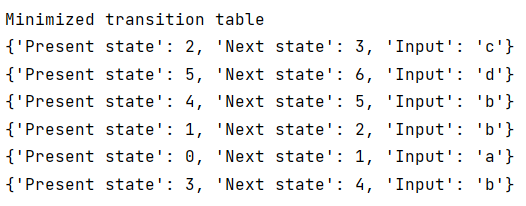


Рисунок 11 – Пример минимизации ДКА для r = abcbdb

**Моделирование КА для входной цепочки из терминалов исходной грамматики**

В данном пункте необходимо реализовать алгоритм моделирования КА для входной цепочки из терминалов исходной грамматики. Примеры валидной и не валидной цепочек для r = (a|b)\*abb представлены на рисунках 12 и 13 соответственно.

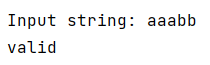


Рисунок 12 – Пример валидной цепочки для r = (a|b)\*abb

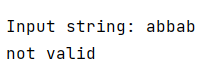


Рисунок 13 – Пример невалидной цепочки для r = (a|b)\*abb

Примеры валидной и не валидной цепочек для r = abcbbd представлены на рисунках 14 и 15 соответственно.

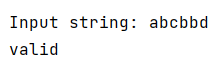


Рисунок 14 – Пример валидной цепочки для r = abcbbd



Рисунок 15 – Пример невалидной цепочки для r = abcbbd

**Текст программы**

run.py

from deterministic\_finite\_automaton import DeterministicFiniteAutomaton  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# regexp = "abcbbd"* regexp = "(a|b)\*abb"  
 marked\_regexp = regexp + "#"  
 print("Start regexp is " + regexp + "\n")  
 print("Marked regexp is " + marked\_regexp + "\n")  
 automaton = DeterministicFiniteAutomaton()  
 automaton.build\_automaton(marked\_regexp)  
 automaton.minimization()  
 check\_string = "aaabb"  
 print("\nInput string: " + check\_string)  
 result = automaton.finite\_automaton\_modeling(check\_string,  
 automaton.start\_state,  
 automaton.state\_transition\_table,  
 automaton.finite\_states)  
 if not result:  
 print("not valid")  
 else:  
 print("valid")

deterministic\_finite\_automaton.py

from node import PosToSymbolMap, FollowPosMap  
from utils import generate\_syntax\_tree  
import numpy as np  
  
  
class DeterministicFiniteAutomaton:  
  
 def \_\_init\_\_(self):  
 self.start\_state = None  
 self.states = None  
 self.state\_transition\_table = None  
 self.all\_symbols = None  
 self.finite\_states = None  
  
 def build\_automaton(self, marked\_regexp):  
 root = generate\_syntax\_tree(marked\_regexp)  
 root.compute()  
 self.start\_state = frozenset(root.firstpos)  
 self.all\_symbols = set(PosToSymbolMap.values()) - set("#")  
 state\_transition\_table = dict()  
 unmrkd\_states = dict()  
 unmrkd\_states[self.start\_state] = False  
  
 while False in unmrkd\_states.values():  
 R = list(unmrkd\_states.keys())[list(unmrkd\_states.values()).index(False)]  
 unmrkd\_states[R] = True  
 for a in self.all\_symbols:  
 S = set()  
 for pos in [p for p in R if PosToSymbolMap[p] == a]:  
 S = S.union(FollowPosMap[pos])  
 if bool(S) and frozenset(S) not in unmrkd\_states:  
 unmrkd\_states[frozenset(S)] = False  
 if R not in state\_transition\_table:  
 state\_transition\_table[R] = dict()  
 state\_transition\_table[R][a] = S  
  
 self.state\_transition\_table = state\_transition\_table  
 self.states = list(unmrkd\_states.keys())  
 pos\_of\_last\_symbol = list(PosToSymbolMap.keys())[list(PosToSymbolMap.values()).index("#")]  
 self.finite\_states = [state for state in self.states if pos\_of\_last\_symbol in state]  
 print("\nStates:")  
 print(self.states)  
  
 print("\nTransition table")  
 for state, num in self.state\_transition\_table.items():  
 print("State:", set(state))  
 for key in num:  
 print(key + ':', num[key])  
  
 def finite\_automaton\_modeling(self, s, state, trans\_table, finite\_states):  
 symbol\_list = list(trans\_table[state])  
 i = 0  
 for ch in s:  
 if not symbol\_list.\_\_contains\_\_(ch):  
 return False  
 if trans\_table.\_\_contains\_\_(state):  
 state = frozenset(trans\_table[state][ch])  
 try:  
 if self.finite\_states.\_\_contains\_\_(state) and len(s) - 1 == i:  
 return True  
 else:  
 symbol\_list = list(trans\_table[state])  
 except:  
 return False  
 i += 1  
 return state in finite\_states  
  
 def get\_position(self, d):  
 if len(d) == 0:  
 *# self.states.index(list(self.states)[len(self.states) - 1])* return len(self.states) - 1  
 return self.states.index(d)  
  
 def minimization(self):  
 n = len(self.states)  
 min\_matrix = np.zeros((n, n))  
 subsets = [list(map(lambda s: self.get\_position(set(s)), filter(lambda s: s not in self.finite\_states, self.states))),  
 list(map(lambda s: self.get\_position(set(s)), self.finite\_states))]  
 for a in subsets[0]:  
 for b in subsets[1]:  
 min\_matrix[a, b] = 1  
 min\_matrix[b, a] = 1  
 is\_finished = False  
  
 while not is\_finished:  
 copy\_min\_matrix = np.copy(min\_matrix)  
 k = 0  
 for i in range(n):  
 for j in range(n):  
 if min\_matrix[i, j] < 1:  
 for e1 in list(list(self.state\_transition\_table.values())[i]):  
 for e2 in list(list(self.state\_transition\_table.values())[j]):  
 if e1 == e2 and min\_matrix[  
 self.get\_position(list(self.state\_transition\_table.values())[i].get(e1)),  
 self.get\_position(list(self.state\_transition\_table.values())[j].get(e2))  
 ] > 0:  
 copy\_min\_matrix[i, j], copy\_min\_matrix[j, i] = 2, 2  
 k += 1  
  
 newSubset = []  
 for i in range(n):  
 f = False  
 newSubSubset = []  
 for j in range(n):  
 if copy\_min\_matrix[i, j] == 0:  
 newSubSubset += [j]  
 if copy\_min\_matrix[i, j] == 2:  
 f = True  
 newSubset += [newSubSubset]  
 f = False  
 newSubset = set(tuple(row) for row in newSubset)  
 min\_matrix = copy\_min\_matrix  
 if k == 0:  
 is\_finished = True  
  
 edges = []  
 for e in newSubset:  
 d = self.state\_transition\_table.get(self.states[e[0]])  
 for k, v in d.items():  
 if len(v) != 0:  
 edges.append({'Present state': e[0], 'Next state': self.get\_position(v), 'Input': k})  
 print("\nMinimized transition table")  
 for edge in edges:  
 print(edge)

utils.py

from node\_types import NodeType  
from node import Node  
  
  
def generate\_syntax\_tree(regexp):  
 root = None  
 left\_set = False  
 for token in tokenize(regexp):  
 if len(token) > 1:  
 node = generate\_syntax\_tree(token)  
 else:  
 node = Node()  
 node.string = token  
 node.node\_type = check\_node\_type(node, token)  
 if root is None:  
 root = node  
 elif left\_set:  
 root.right = node  
 left\_set = False  
 else:  
 node.left = root  
 if node.node\_type != NodeType.Star:  
 left\_set = True  
 root = node  
 return root  
  
  
def tokenize(regexp):  
 tokens = []  
 i = 0  
 is\_multiplier = False  
 while i < len(regexp):  
 ch = regexp[i]  
 if ch == "(":  
 token, i = process\_brackets(regexp, i)  
 if is\_multiplier:  
 tokens.append(".")  
 tokens.append(token)  
 else:  
 if ch in "\*|":  
 tokens.append(ch)  
 else:  
 if is\_multiplier:  
 tokens.append(".")  
 tokens.append(ch)  
 i += 1  
 is\_multiplier = ch != "|"  
 return tokens  
  
  
def process\_brackets(regexp, i):  
 brackets\_counter = 1  
 j = i  
 while brackets\_counter != 0 and j < len(regexp):  
 j += 1  
 ch = regexp[j]  
 if ch == "(":  
 brackets\_counter += 1  
 elif ch == ")":  
 brackets\_counter -= 1  
 return regexp[i + 1:j], j  
  
  
def check\_node\_type(node, token):  
 if token == "\*":  
 node.node\_type = NodeType.Star  
 elif token == "|":  
 node.node\_type = NodeType.Or  
 elif token == ".":  
 node.node\_type = NodeType.Concatenation  
 else:  
 node.node\_type = NodeType.Symbol  
 return node.node\_type

node.py

from node\_types import NodeType  
  
PosToSymbolMap = dict()  
FollowPosMap = dict()  
  
  
class Node:  
 def \_\_init\_\_(self):  
 self.string = ""  
 self.node\_type = None  
 self.left = None  
 self.right = None  
 self.pos = None  
 self.nullable = None  
 self.firstpos = []  
 self.lastpos = []  
  
 def assign\_numbers\_to\_symbols(self, i):  
 if self.string not in "\*|.":  
 self.pos = i + 1  
 PosToSymbolMap[self.pos] = self.string  
 return self.pos  
 else:  
 left\_pos = self.left.assign\_numbers\_to\_symbols(i)  
 if self.string == "\*":  
 result = left\_pos  
 else:  
 result = self.right.assign\_numbers\_to\_symbols(left\_pos)  
 return result  
  
 def compute\_null\_first\_last(self):  
 if self.pos is not None:  
 self.nullable = False  
 self.firstpos.append(self.pos)  
 self.lastpos.append(self.pos)  
 elif self.node\_type == NodeType.Star:  
 self.nullable = True  
 self.left.compute\_null\_first\_last()  
 self.firstpos = self.left.firstpos  
 self.lastpos = self.left.lastpos  
 else:  
 self.left.compute\_null\_first\_last()  
 self.right.compute\_null\_first\_last()  
 self.firstpos.extend(self.left.firstpos)  
 self.lastpos.extend(self.right.lastpos)  
 if self.node\_type == NodeType.Or:  
 self.nullable = self.left.nullable or self.right.nullable  
 self.firstpos.extend(self.right.firstpos)  
 self.lastpos.extend(self.left.lastpos)  
 elif self.node\_type == NodeType.Concatenation:  
 self.nullable = self.left.nullable and self.right.nullable  
 if self.left.nullable:  
 self.firstpos.extend(self.right.firstpos)  
 if self.right.nullable:  
 self.lastpos.extend(self.left.lastpos)  
  
 def compute\_follow(self, i):  
 result = []  
 if self.node\_type == NodeType.Concatenation and i in self.left.lastpos:  
 result = self.right.firstpos  
 elif self.node\_type == NodeType.Star and i in self.lastpos:  
 result = self.left.firstpos  
  
 if self.left is not None:  
 result.extend(self.left.compute\_follow(i))  
 if self.right is not None:  
 result.extend(self.right.compute\_follow(i))  
  
 return set(result)  
  
 def compute(self):  
 self.assign\_numbers\_to\_symbols(0)  
 print("Numbers for each symbols in tree:")  
 print(PosToSymbolMap)  
  
 self.compute\_null\_first\_last()  
 for key in PosToSymbolMap.keys():  
 FollowPosMap[key] = self.compute\_follow(key)  
  
 print("\nFollowpos:")  
 print(FollowPosMap)

node\_types.py

class NodeType:  
 Empty = 0  
 Symbol = 1  
 Concatenation = 2  
 Star = 3  
 Or = 4

**Список литературы**

##### [1] Citforum [Электронный ресурс] // Построение детерминированного конечного автомата по регулярному выражению, URL: <http://citforum.ru/programming/theory/serebryakov/3.shtml>

##### [2] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий, 2-е издание. И.Д. Вильямс, 2008 – 1184 с.

##### [3] Enppt [Электронный ресурс] // Автоматы и формальные языки, URL: <https://en.ppt-online.org/40054>