Государственное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»

Отчет
По лабораторной работе
По курсу «Конструирование компиляторов»
На тему
«Синтаксический разбор с использованием метода рекурсивного спуска»

Студент: Горин Д.И.
Группа: ИУ7-23М
Вариант: 3
Преподаватель: Ступников А.А.

Москва, 2020

Оглавление

1 Цель и задачи работы

Цель работы: Дополнить грамматику блоком, состоящим из последовательности операторов присваивания. Для реализации предлагаются два варианта расширенной грамматики.

2 Листинг

2.1 main.py

```
from grammar import Grammar
from ll import build_tree, graph_tree


if __name__ == '__main__':
    g = Grammar.init_from_json_file('grammar_g3_no_lrec_c.json')
    print(g, end='\n\n')

    initial = g.initial_nterm

    expr = '{x=2<2}'
    print(expr)
    is_ok, _ = build_tree(grammar=g, current_symbol=initial,
        string_to_read=expr)
    if is_ok is not None:
        graph_tree(is_ok, filename='2l2')
    print(f'{is_ok is not None}, должно быть True\n')

    expr = '{x=2*3>=2/4}'
    print(expr)
    is_ok, _ = build_tree(grammar=g, current_symbol=initial,
        string_to_read=expr)
    if is_ok is not None:
        graph_tree(is_ok, filename='2m3ge2d4')
    print(f'{is_ok is not None}, должно быть True\n')

    expr = '{x=(2+3)<2}'
    print(expr)
    is_ok, _ = build_tree(grammar=g, current_symbol=initial,
        string_to_read=expr)
    if is_ok is not None:
        graph_tree(is_ok, filename='p2p3pl2')
    print(f'{is_ok is not None}, должно быть True\n')

    expr = '{x=2===2}'
    print(expr)
    is_ok, _ = build_tree(grammar=g, current_symbol=initial,
        string_to_read=expr)
    if is_ok is not None:
        graph_tree(is_ok, filename='2eee2')
```

```
37      print(f'{is_ok is not None}, должно быть False\n')
38
39      expr = '{x=2=3/4==2}'
40      print(expr)
41      is_ok, _ = build_tree(grammar=g, current_symbol=initial,
            string_to_read=expr)
42      if is_ok is not None:
43          graph_tree(is_ok, filename='2a3d5ee2')
44      print(f'{is_ok is not None}, должно быть False\n')
```

2.2 ll.py

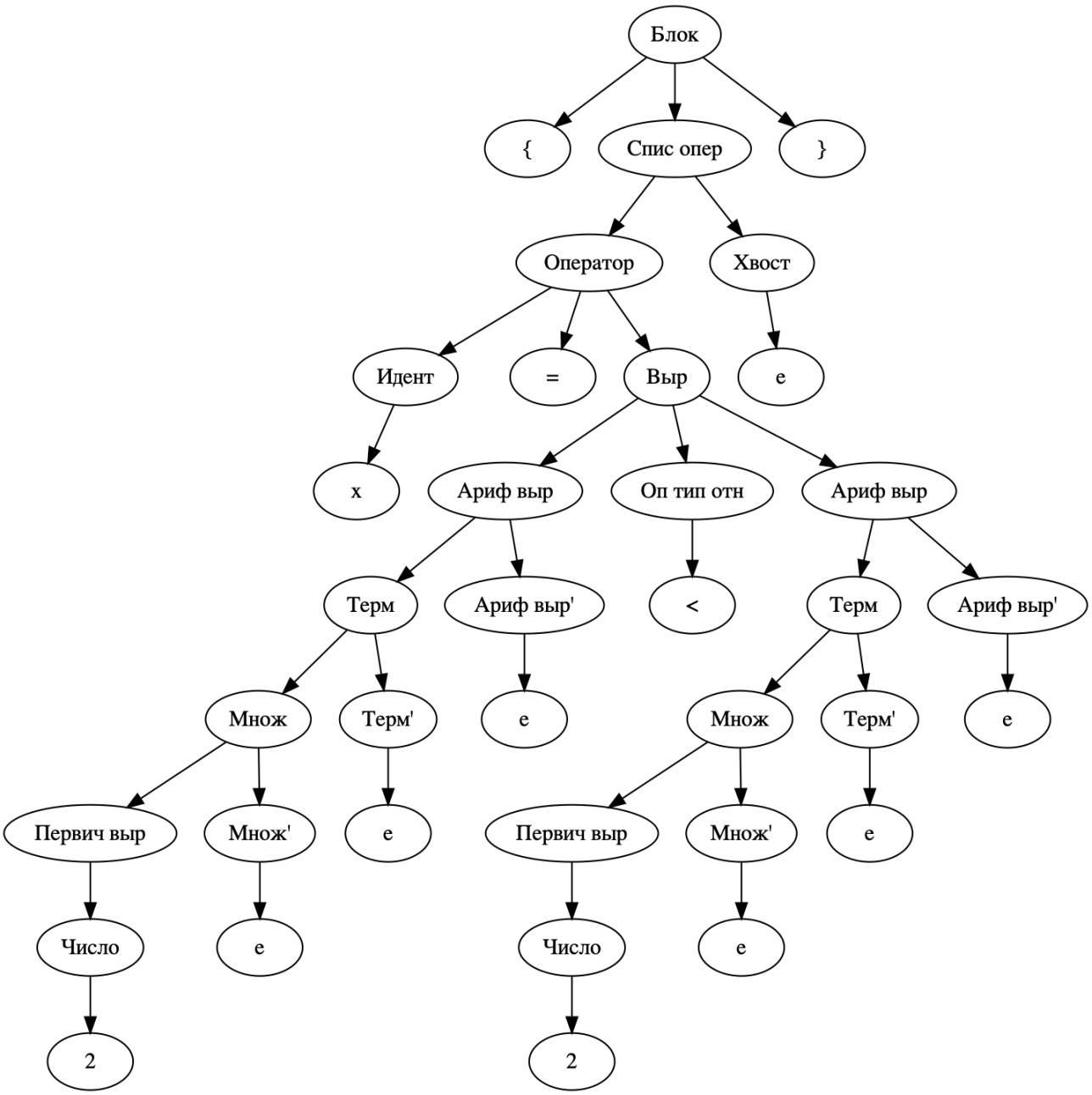
```
1  from graphviz import Digraph
2  from grammar import Grammar, NoTermSymbol, TermSymbol
3
4
5  # def build_tree(grammar: Grammar, current_symbol, string_to_read)
       -> (bool, str):
6  #     flag = False
7  #     new_str = string_to_read
8  #     for production in grammar.rules[current_symbol]:
9  #         for prod_sym in production:
10 #             if isinstance(prod_sym, NoTermSymbol):
11 #                 flag, new_str = build_tree(grammar, prod_sym,
       new_str)
12 #             else:
13 #                 cur_term_sym = prod_sym.symbol
14 #                 if cur_term_sym == grammar.eps_terminal.symbol:
15 #                     flag = True
16 #                 elif cur_term_sym == new_str[:len(cur_term_sym)]:
17 #                     flag = True
18 #                     new_str = new_str[len(cur_term_sym):]
19 #                 else:
20 #                     flag = False
21 #             if not flag:
22 #                 break
23 #         if flag:
24 #             break
25 #     return flag, new_str
26
27
28 class TreeNode:
29      def __init__(self, value: str, children: list = None):
30          self.value = value
31          self.children = children or []
32
33      def clear_children(self):
34          self.children = []
35
36      def append_child(self, node: 'TreeNode'):
37          self.children.append(node)
```

```
38
39      def __str__(self):
40          return f'{self.value} -> {self.children}'
41
42      def __repr__(self):
43          return str(self)
44
45
46  def build_tree(grammar: Grammar, current_symbol, string_to_read) ->
        (TreeNode, str):
47      if isinstance(current_symbol, TermSymbol):  # Еслитерминал
48          if current_symbol == grammar.eps_terminal:  # Еслиэпссимвол -,
                топростовозвращаемего
49              return TreeNode(current_symbol.symbol), string_to_read
50          elif current_symbol.symbol == string_to_read[:len(
                current_symbol)]:  # Есливначалестрокиестьтерминал , тоок
51              return TreeNode(current_symbol.symbol), string_to_read[
                    len(current_symbol):]
52          else:  # Иначеошибка , возвращаем None
53              return None, string_to_read
54      else:  # Еслинетерминал
55          for rule in grammar.rules[current_symbol]:  #
                Покаждомуправилуизграмматикипопереданномунетерминалу
56              new_str = string_to_read
57              ret_node = TreeNode(current_symbol.symbol)
58              for symbol in rule:  # Покаждомусимволувправиле
59                  symbol_children, new_str = build_tree(grammar,
                        symbol, new_str)
60                  if symbol_children is None:  #
                        Еслихотябыпоодномусимволуошибка —
                        правилонеподходит
61                      break
62                  ret_node.append_child(symbol_children)
63              else:  # Есливышлибезбряка , тонашлинужноеподдерево
64                  return ret_node, new_str
65      return None, string_to_read  # Еслиненашлосьверногоправила ,
            значитошибка
66
67
68  def graph_tree(root_node: TreeNode, filename='tree'):
69      d = Digraph()
70      cnt = 0
71      stack = [(root_node, cnt)]
72      while len(stack) > 0:
73          node, num = stack.pop()
74          d.node(node.value + str(num), label=node.value)
75          for child in node.children:
76              cnt += 1
77              stack.append((child, cnt))
78              d.edge(node.value + str(num), child.value + str(cnt))
```

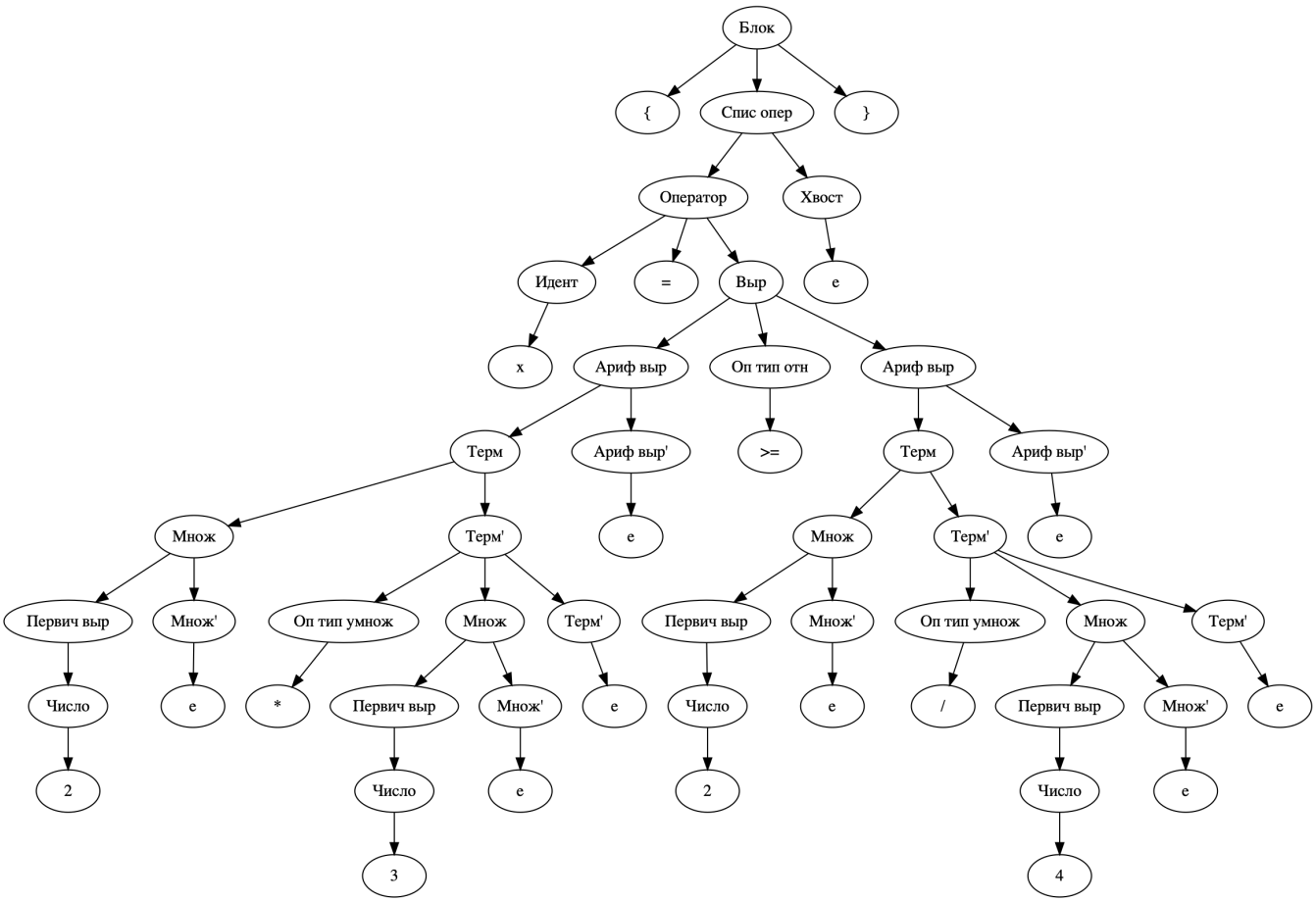
```
79      d.save(filename)
80      d.render(filename, view=True)
```

3 Тесты

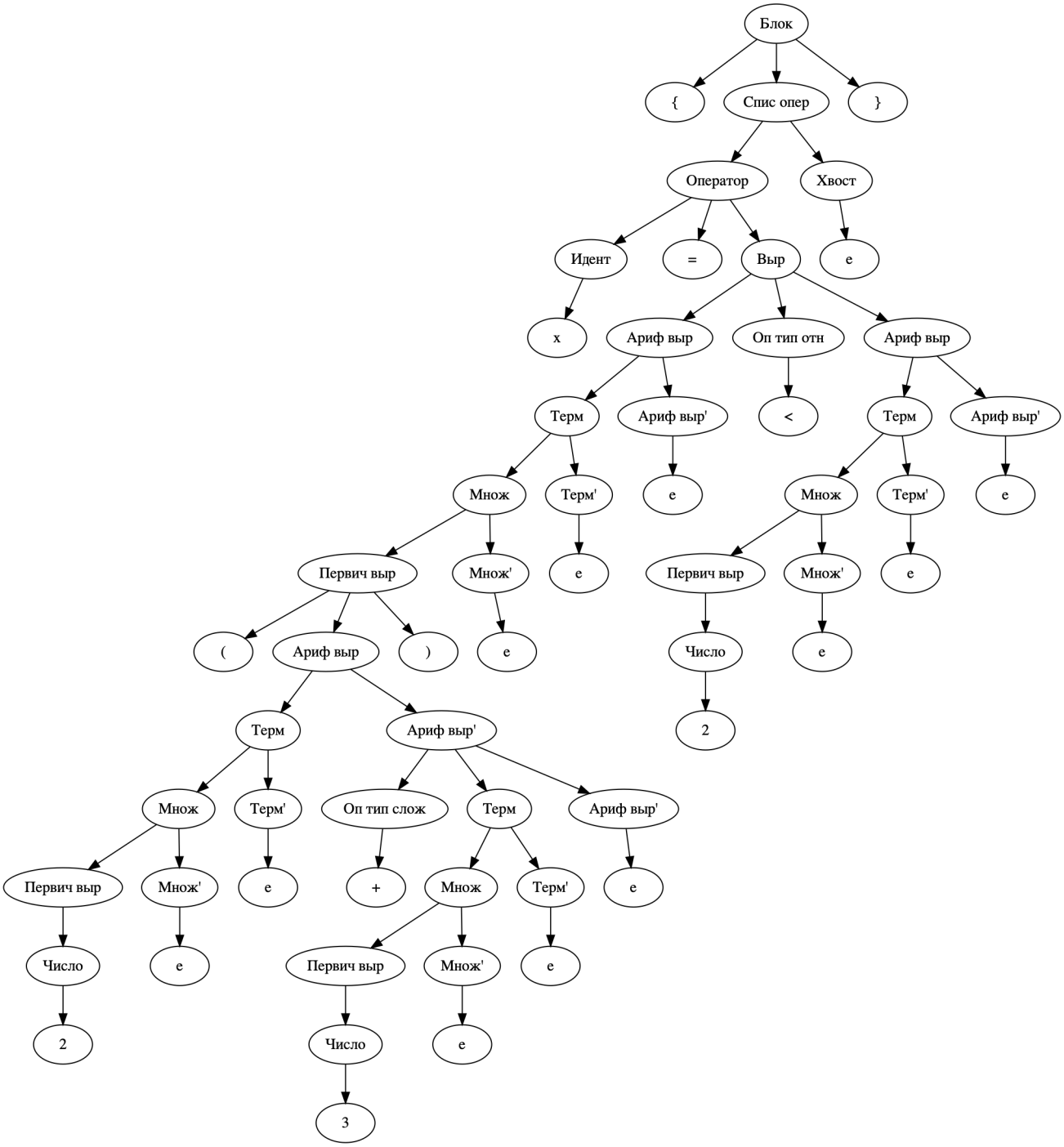
3.1 $\{x = 2 == 2\}$



3.2 $\{x = 2 * 3 >= 2/4\}$


Блок
{
Спис опер
}
Оператор
Хвост
Идент
=
Выр
e
x
Ариф выр
Оп тип отн
Ариф выр
Терм
Ариф выр'
>=
Терм
Ариф выр'
Множ
Терм'
e
Множ
Терм'
e
Первич выр
Множ'
Оп тип умнож
Множ
Терм'
Первич выр
Множ'
Оп тип умнож
Множ
Терм'
Число
e
*
Первич выр
Множ'
e
Число
e
/
Первич выр
Множ'
e
2
Число
e
2
Число
e
3
4

3.3 $\{x = (2 + 3) < 2\}$



3.4 $\{x = 2 === 2\}$

Дерево не построится – ошибка входной строки

3.5 $\{x = 2 = 3/4 == 2\}$

Дерево не построится – ошибка входной строки

4 Выводы

По результатам проведенной работы студент приобрел практические навыки в реализации алгоритма синтаксического разбора с использованием рекурсивного спуска

5 Список литературы

1. БЕЛОУСОВ А.И., ТКАЧЕВ С.Б. Дискретная математика: Учеб. Для вузов / Под ред. В.С. Зарубина, А.П. Крищенко. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.

2. АХО А., УЛЬМАН Дж. Теория синтаксического анализа, перевода и компиляции: В 2-х томах. Т.1.: Синтаксичечкий анализ. - М.: Мир, 1978.

3. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.