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 (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

ОТЧЕТ

Лабораторная работа №2

по курсу «Конструирование компиляторов»

на тему: «Преобразования грамматик»

Вариант № 6

Студент ИУ7-22М (Группа) _____ (Подпись, дата) И. А. Цветков (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) А. А. Ступников (И. О. Фамилия)

2024 г.

1 Задание

1.1 Задание

1.2 Общий вариант для всех: Устранение левой рекурсии.

Определение. Нетерминал A КС-грамматики $G = (N, \Sigma, P, S)$ называется рекурсивыным, если $A => +\alpha A \beta$ для некоторых α и β. Если $\alpha = \epsilon$, то A называется леворекурсивным. Аналогично, если $\beta = \epsilon$, то называется праворекурсивным. Грамматика, имеющая хотя бы один леворекурсивный нетерминал, называется леворекурсивной. Аналогично определяется праворекурсивная грамматика. Грамматика, в которой все нетерминалы, кроме, быть может, начального символа, рекурсивные, называется рекурсивной.

Некоторые из алгоритмов разбора не могут работать с леворекурсивными грамматиками. Можно показать, что каждый КС-язык определяется хотя бы одной не леворекурсивной грамматикой.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику G' без левой рекурсии.

1.3 Преобразование к нормальной форме Хомского.

Определение. КС-грамматика $G = (N, \Sigma, P, S)$ называется грамматикой в `нормальной форме Хомского` (или в `бинарной нормальной форме`), если каждое правило из имеет один из следующих видов:

1. $A \to BC$, где A, B и C принадлежат N,
2. $A \to a$, где $a \in \Sigma$,
3. $S \to \epsilon$, если $\epsilon \in L(G)$, причем S не встречается в правых частях правил.

Можно показать, что каждый КС-язык порождается грамматикой в нормальной форме Хомского. Этот результат полезен в случаях, когда требуется простая форма представления КС-языка.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику G' в нормальной форме Хомского.

2 Выполнение лабораторной работы

Результаты работы программы по преобразованию грамматик приведены на рисунках 2.1–2.14.

2.0.1 Устранение левой рекурсии

```
G = ({E, T, F}, {+, *, (, ), a}, P, E), где P состоит из правил:

E -> E + T | T
T -> T * F | F
F -> a | ( E )
```

Рисунок 2.1 – Исходная грамматика для удаления левой рекурсии (пример 1)

```
G = ({E, E', T, T', F}, {+, *, (, ), a}, P, E), где P состоит из правил:

E -> T E'
E' -> + T E' | Ɛ
T -> F T'
T' -> * F T' | Ɛ
F -> a | ( E )
```

Рисунок 2.2 – Грамматика после удаления левой рекурсии (пример 1)

```
G = ({S, A}, {a, b, c, d}, P, A), где P состоит из правил:

S -> A a | b
A -> A c | S d | Ɛ
```

Рисунок 2.3 – Исходная грамматика для удаления левой рекурсии (пример 2)

```
G = ({S, A, A'}, {a, b, c, d}, P, A), где P состоит из правил:

S -> A a | b
A -> b d A' | A'
A' -> c A' | a d A' | Ɛ
```

Рисунок 2.4 – Грамматика после удаления левой рекурсии (пример 2)

```
G = ({E, T, F}, {+, -, *, /, (, ), id}, P, E), где P состоит из правил:

E -> E + T | E - T | T
T -> T * F | T / F | F
F -> ( E ) id
```

Рисунок 2.5 – Исходная грамматика для удаления левой рекурсии (пример 3)

```
G = ({E, E', T, T', F}, {+, -, *, /, (, ), id}, P, E), где P состоит из правил:

E -> T E'
E' -> + T E' | - T E' | Ɛ
T -> F T'
T' -> * F T' | / F T' | Ɛ
F -> ( E ) id
```

Рисунок 2.6 – Грамматика после удаления левой рекурсии (пример 3)

2.0.2 Устранение левой факторизации

```
G = ({S, E}, {i, t, e, a, b}, P, S), где P состоит из правил:

S -> i E t S | i E t S e S | a
E -> b
```

Рисунок 2.7 – Исходная грамматика для удаления левой факторизации

```
G = ({S, S', E}, {i, t, e, a, b}, P, S), где P состоит из правил:

S -> i E t S S' | a
S' -> Ɛ | e S
E -> b
```

Рисунок 2.8 – Грамматика после удаления левой факторизации

2.0.3 Преобразование КС-грамматики к нормальной форме Хомского

```
G = ({S, A, B}, {a, b}, P, S), где P состоит из правил:

S -> a A B | B A
A -> B B B | a
B -> A S | b
```

Рисунок 2.9 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 1)

```
G = ({S, A, B, a', <AB>, <BB>}, {a, b}, P, S), где P состоит из правил:

S -> a' <AB> | B A
A -> B <BB> | a
B -> A S | b
a' -> a
<AB> -> A B
<BB> -> B B
```

Рисунок 2.10 – Грамматика после преобразования к нормальной форме Хомского (пример 1)

```
G = ({S}, {0, 1}, P, S), где P состоит из правил:

S -> 0 S 1 | 0 1
```

Рисунок 2.11 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 2)

```
G = ({S, 0', <S1>, 1'}, {0, 1}, P, S), где P состоит из правил:

S -> 0' <S1> | 0' 1'
0' -> 0
<S1> -> S 1'
1' -> 1
```

Рисунок 2.12 – Грамматика после преобразования к нормальной форме Хомского (пример 2)

```
G = ({S, A, B}, {a, b}, P, S), где P состоит из правил:

S -> a B | b A
A -> a S | b A A | a
B -> b S | a B B | b
```

Рисунок 2.13 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 3)

```
G = ({S, A, B, a', b', <AA>, <BB>}, {a, b}, P, S), где P состоит из правил:

S -> a' B | b' A
A -> a' S | b' <AA> | a
B -> b' S | a' <BB> | b
a' -> a
b' -> b
<AA> -> A A
<BB> -> B B
```

Рисунок 2.14 – Грамматика после преобразования к нормальной форме Хомского (пример 3)

2.1 Контрольные вопросы

1. Как может быть определён формальный язык?
   (a) Простым перечислением слов, входящих в данный язык.
   (b) Словами, порождёнными некоторой формальной грамматикой.
   (c) Словами, порождёнными регулярным выражением.
   (d) Словами, распознаваемыми некоторым конечным автоматом.
2. Какими характеристиками определяется грамматика?
   (a) Σ — множество терминальных символов.
   (b) N — множество нетерминальных символов.
   (c) P — множество правил (слева — непустая последовательность терминалов/нетерминалов, содержащая хотя бы один нетерминал, справа — любая последовательность терминалов/нетерминалов).
   (d) S — начальный символ из множества нетерминалов.
3. Дайте описания грамматик по иерархии Хомского.
   (a) Регулярные.
   (b) Контекстно-свободные.
   (c) Контекстно-зависимые.
   (d) Неограниченные.
4. Какие абстрактные устройства используются для разбора грамматик?
   (a) Распознающие грамматики — устройства (алгоритмы), которым на вход подается цепочка языка, а на выходе устройство печатает «Да», если цепочка принадлежит языку, и «Нет» — иначе.
5. Оцените временную и емкостную сложность предложенного вам алгоритма.
   (a) Алгоритм удаления левой рекурсии
      — $O(N^2)$ — временная сложность;
      — $O(N)$ — ёмкостная сложность.

2.2 Код прогаммы

В листингах 2.1–2.2 представлен код программы.

Листинг 2.1 — Основной модуль программы

```
import subprocess

from grammar import Grammar, reedGrammarFromFile


SIZE_MENU = 5
OUTPUT_FILE_NAME = "./data/result.txt"
MENU = f"""
  \tМеню\n
  1.  Вывести исходную грамматику
  2.  Грамматика после устранения левой рекурсии
  3.  Грамматика после устранения левой факторизации
  4.  Грамматика после устранения левой рекурсии и левой факторизации
  5.  Преобразование КС-грамматики к нормальной форме Хомского

  0.  Выход\n
  Выбор: """


def inputOption(minOptions: int, maxOptions: int, msg: str):
  try:
    option = int(input(msg))
  except:
    option = -1
  else:
    if option < minOptions or option > maxOptions:
      option = -1

  if option == -1:
    print(f"\nОжидался ввод целого числа от {minOptions} до {maxOptions}")

  return option


def chooseInputFile() -> str:
  with open("temp.txt", "w") as f:
    subprocess.run(["ls", "./data"], stdout=f)

  with open("temp.txt") as f:
    fileNames = [line[:-1] for line in f.readlines()]

  subprocess.run(["rm", "temp.txt"])
```

Продолжение листинга 2.1

```
43
44      msg = f"\n\tВходные файлы:\n\n"
45      for i in range(len(fileNames)):
46        msg += f"  {i + 1}.  {fileNames[i]};\n"
47      msg += f"\n  Выбор: "
48
49      option = -1
50      while option == -1:
51        option = option = inputOption(
52          minOptions=1,
53          maxOptions=len(fileNames),
54          msg=msg,
55        )
56
57      return f"./data/{fileNames[option - 1]}"
58
59
60    def main():
61      inputFile = chooseInputFile()
62      option = -1
63      while option != 0:
64        option = inputOption(
65          minOptions=0,
66          maxOptions=SIZE_MENU,
67          msg=MENU,
68        )
69        match option:
70          case 1:
71            grammar: Grammar = reedGrammarFromFile(inputFile)
72            grammar.printGrammar()
73          case 2:
74            grammar: Grammar = reedGrammarFromFile(inputFile)
75            grammar.removeLeftRecursion()
76            grammar.printGrammar()
77            grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
78          case 3:
79            grammar: Grammar = reedGrammarFromFile(inputFile)
80            grammar.removeLeftFactorization()
81            grammar.printGrammar()
82            grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
83          case 4:
84            grammar: Grammar = reedGrammarFromFile(inputFile)
85            grammar.removeLeftRecursion()
86            grammar.removeLeftFactorization()
87            grammar.printGrammar()
```

Продолжение листинга 2.1

```
88            grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
89          case 5:
90            grammar: Grammar = reedGrammarFromFile(inputFile)
91            grammar.convertToChomskyForm()
92            grammar.printGrammar()
93            grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
94
95
96  if __name__ == '__main__':
97    main()
```

Листинг 2.2 — Модуль для преобразования грамматик

```
1   from functools import reduce
2   from copy import deepcopy
3
4
5   class Grammar:
6     notTerminals: list[str]
7     terminals: list[str]
8     rules: dict[str, list[list[str]]]
9     start: str
10
11    def __init__(
12      self,
13      notTerminals: list[str],
14      terminals: list[str],
15      rules: dict[str, list[list[str]]],
16      start: str
17    ) -> None:
18      self.notTerminals = notTerminals
19      self.terminals = terminals
20      self.rules = rules
21      self.start = start
22
23    def printGrammar(self) -> None:
24      notTerminals = Grammar.__joinListWithSymbol(self.notTerminals, ", ")
25      terminals = Grammar.__joinListWithSymbol(self.terminals, ", ")
26
27      print(f"\nG = ({{{notTerminals}}}, {{{terminals}}}, P, {self.start}), где P
        ↪  состоит из правил:\n")
28      for  notTerminal in self.notTerminals:
29        rightRules = self.rules[notTerminal]
30        self.__printProduct(notTerminal, rightRules)
31
```

Продолжение листинга 2.2

```
32      def removeLeftRecursion(self) -> None:
33        i = 0
34        while i < len(self.notTerminals):
35          copyRightRules = self.rules[self.notTerminals[i]].copy()
36          for j in range(i):
37            self.__replaceProducts(
38              notTerminal=self.notTerminals[i],
39              replaceableNotTerminal=self.notTerminals[j],
40            )
41          if self.__removeDirectLeftRecursion(self.notTerminals[i]):
42            i += 2
43          else:
44            self.rules[self.notTerminals[i]] = copyRightRules
45            i += 1
46
47      def removeLeftFactorization(self) -> None:
48        i = 0
49        while i < len(self.notTerminals):
50          maxPrefix = ""
51          rightRules = self.rules[self.notTerminals[i]]
52          for j in range(len(rightRules)):
53            prefix = ""
54            for symbol in rightRules[j]:
55              indexList = self.__findPrefixMatches(
56                rightRules=rightRules,
57                prefix=prefix + symbol,
58              )
59              if len(indexList) > 1:
60                prefix += symbol
61              else:
62                break
63
64            if len(prefix) > len(maxPrefix):
65              maxPrefix = prefix
66
67          if maxPrefix:
68            print(f"\nСамый длинный префикс для {self.notTerminals[i]}: {maxPrefix}")
69            self.__removeDirectLeftFactorization(self.notTerminals[i], maxPrefix)
70          else:
71            i += 1
72
73      def convertToChomskyForm(self) -> None:
74        for notTerminal in self.notTerminals.copy():
75          for i in range(len(self.rules[notTerminal])):
76            rightRule = self.rules[notTerminal][i]
```

Продолжение листинга 2.2

```
77            if len(rightRule) == 2 and \
78              rightRule[0] in self.notTerminals and \
79              rightRule[1] in self.notTerminals or \
80              len(rightRule) == 1 and rightRule[0] in self.terminals or \
81              notTerminal == self.start and rightRule[0] == "e":
82              continue
83
84            elif len(rightRule) == 2 and \
85              (rightRule[0] in self.terminals or rightRule[1] in self.terminals):
86              if rightRule[0] in self.terminals:
87                firstElem = f"{rightRule[0]}'"
88                if not firstElem in self.notTerminals:
89                  self.notTerminals.append(firstElem)
90                  self.rules[firstElem] = [[rightRule[0]]]
91              else:
92                firstElem = rightRule[0]
93
94              if rightRule[1] in self.terminals:
95                secondElem = f"{rightRule[1]}'"
96                if not secondElem in self.notTerminals:
97                  self.notTerminals.append(secondElem)
98                  self.rules[secondElem] = [[rightRule[1]]]
99              else:
100               secondElem = rightRule[1]
101
102             self.rules[notTerminal][i] = [firstElem, secondElem]
103
104           elif len(rightRule) > 2:
105             if rightRule[0] in self.notTerminals:
106               self.rules[notTerminal][i] = [rightRule[0],
                  ↪  f"<{''.join(rightRule[1:])}>"]
107             else:
108               self.rules[notTerminal][i] = [f"{rightRule[0]}'",
                  ↪  f"<{''.join(rightRule[1:])}>"]
109               if not f"{rightRule[0]}'" in self.notTerminals:
110                 self.notTerminals.append(f"{rightRule[0]}'")
111                 self.rules[f"{rightRule[0]}'"] = [[rightRule[0]]]
112
113             rightRule = rightRule[1:]
114             newNotTerminal = f"<{''.join(rightRule)}>"
115             while len(rightRule) > 2:
116               if not newNotTerminal in self.notTerminals:
117                 self.notTerminals.append(newNotTerminal)
118
119                 if rightRule[0] in self.notTerminals:
```

Продолжение листинга 2.2

```
                    self.rules[newNotTerminal] = [[rightRule[0],
                    ↪  f"<{''.join(rightRule[1:])}>"]]
                  else:
                    self.rules[newNotTerminal] = [[f"{rightRule[0]}'",
                    ↪  f"<{''.join(rightRule[1:])}>"]]
                    if not f"{rightRule[0]}'" in self.notTerminals:
                      self.notTerminals.append(f"{rightRule[0]}'")
                      self.rules[f"{rightRule[0]}'"] = [[rightRule[0]]]

                rightRule = rightRule[1:]
                newNotTerminal = f"<{''.join(rightRule)}>"

              newNotTerminal = f"<{''.join(rightRule)}>"
              if not newNotTerminal in self.notTerminals:
                if rightRule[0] in self.terminals:
                  firstElem = f"{rightRule[0]}'"
                  if not firstElem in self.notTerminals:
                    self.notTerminals.append(firstElem)
                    self.rules[firstElem] = [[rightRule[0]]]
                else:
                  firstElem = rightRule[0]

                if rightRule[1] in self.terminals:
                  secondElem = f"{rightRule[1]}'"
                  if not secondElem in self.notTerminals:
                    self.notTerminals.append(secondElem)
                    self.rules[secondElem] = [[rightRule[1]]]
                else:
                  secondElem = rightRule[1]

                self.notTerminals.append(newNotTerminal)
                self.rules[newNotTerminal] = [[firstElem, secondElem]]

    def createFileFromGrammar(self, fileName: str) -> None:
      with open(fileName, "w") as f:
        for i in range(len(self.notTerminals)):
          if i:
            f.write(" ")
          f.write(f"{self.notTerminals[i]}")
        f.write("\n")

        for i in range(len(self.terminals)):
          if i:
            f.write(" ")
          f.write(f"{self.terminals[i]}")
```

Продолжение листинга 2.2

```
163         f.write("\n")
164
165         for notTerminal in self.notTerminals:
166           for rightRule in self.rules[notTerminal]:
167             f.write(f"{notTerminal} ->")
168             for symbol in rightRule:
169               f.write(f" {symbol}")
170             f.write("\n")
171
172         f.write(f"{self.start}\n")
173
174     def __removeDirectLeftFactorization(self, notTerminal: str, maxPrefix: str) ->
    ↪  None:
175       indexList = self.__findPrefixMatches(
176         rightRules=self.rules[notTerminal],
177         prefix=maxPrefix,
178       )
179       newRightRules = []
180       lenMaxPrefix = len(maxPrefix)
181       for i in indexList:
182         if len(self.rules[notTerminal][i]) > lenMaxPrefix:
183           newRightRules.append(self.rules[notTerminal][i][lenMaxPrefix:])
184         else:
185           newRightRules.append(["e"])
186
187       rightRules = []
188       for i in range(len(self.rules[notTerminal])):
189         if not i in indexList:
190           rightRules.append(self.rules[notTerminal][i])
191
192       newNotTerminal = self.__findNewNotTerminal(notTerminal)
193       self.rules[newNotTerminal] = newRightRules
194       self.rules[notTerminal] = \
195         [list(maxPrefix) + [newNotTerminal]] + rightRules
196
197       indexNotTerminal = self.notTerminals.index(notTerminal)
198       self.notTerminals = \
199         self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
200         self.notTerminals[indexNotTerminal + 1:]
201
202     def __findNewNotTerminal(self, notTerminal: str):
203       try:
204         baseNotTerminal = notTerminal[:notTerminal.index("'")]
205       except ValueError:
206         baseNotTerminal = notTerminal
```

Продолжение листинга 2.2

```
207
208       quotationMarkCount = 0
209       for item in self.notTerminals:
210         if item.find(baseNotTerminal) != -1:
211           quotationMarkCount += 1
212
213       return notTerminal + "'" * quotationMarkCount
214
215     def __findPrefixMatches(self, rightRules: list[list[str]], prefix: str) ->
      ↪  list[int]:
216       indexList = []
217       for i in range(len(rightRules)):
218         if self.__comparePrefixes(rightRules[i], prefix):
219           indexList.append(i)
220
221       return indexList
222
223     def __comparePrefixes(self, rightRule: list[str], prefix: str) -> bool:
224       if len(rightRule) < len(prefix):
225         return False
226
227       for i in range(len(prefix)):
228         if prefix[i] != rightRule[i]:
229           return False
230
231       return True
232
233     def __replaceProducts(self, notTerminal: str, replaceableNotTerminal: str) ->
      ↪  None:
234       flagReplace = False
235       newRightRules = []
236       rightRules = self.rules[notTerminal]
237       for i in range(len(rightRules)):
238         if replaceableNotTerminal not in rightRules[i]:
239           newRightRules.append(rightRules[i])
240           continue
241
242         flagReplace = True
243         j = rightRules[i].index(replaceableNotTerminal)
244         for substitutedRightRule in self.rules[replaceableNotTerminal]:
245           newRightRule = rightRules[i][:j]
246           if substitutedRightRule[0] != "e":
247             newRightRule.extend(substitutedRightRule)
248           newRightRule.extend(rightRules[i][j + 1:])
249           newRightRules.append(newRightRule)
```

Продолжение листинга 2.2

```
250
251       if flagReplace:
252         self.rules[notTerminal] = newRightRules
253         print(f"\nПосле замены {replaceableNotTerminal}: ", end="")
254         self.__printProduct(notTerminal, newRightRules)
255
256     def __removeDirectLeftRecursion(self, notTerminal: str) -> bool:
257       self.rules[notTerminal].sort(
258         key=lambda rightRule: rightRule[0] != notTerminal
259       )
260       newNotTerminal = notTerminal + "'"
261       rightRulesForNewNotTerminal = []
262       rightRules = []
263
264       for rightRule in deepcopy(self.rules[notTerminal]):
265         if rightRule[0] != notTerminal:
266           if rightRule[0] == "e":
267             rightRule = [newNotTerminal]
268           else:
269             rightRule.append(newNotTerminal)
270           rightRules.append(rightRule)
271         else:
272           rightRule = rightRule[1:]
273           rightRule.append(newNotTerminal)
274           rightRulesForNewNotTerminal.append(rightRule)
275
276       if len(rightRulesForNewNotTerminal):
277         rightRulesForNewNotTerminal.append(["e"])
278         indexNotTerminal = self.notTerminals.index(notTerminal)
279         self.notTerminals = \
280           self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
281           self.notTerminals[indexNotTerminal + 1:]
282         self.rules[newNotTerminal] = rightRulesForNewNotTerminal
283         self.rules[notTerminal] = rightRules
284
285         removedFlag = True
286       else:
287         removedFlag = False
288
289       return removedFlag
290
291     def __printProduct(self, notTerminal: str, rightRules: list[list[str]]):
292       print(f"{notTerminal} -> ", end="")
293       for i in range(len(rightRules)):
```

Продолжение листинга 2.2

```
294         print(f"{' | ' if i != 0 else ''}{Grammar.__joinListWithSymbol(rightRules[i],
            ↪  ' ')}", end="")
295       print()
296
297     @staticmethod
298     def __joinListWithSymbol(arr: list[str], symbol: str) -> str:
299       return reduce(lambda elemPrev, elem: f"{elemPrev}{symbol}{elem}", arr)
300
301
302   def reedGrammarFromFile(fileName: str) -> Grammar:
303     with open(fileName) as f:
304       lines = [line[:-1] for line in f.readlines()]
305
306     notTerminals = lines[0].split(" ")
307     terminals = lines[1].split(" ")
308     start = lines[-1]
309     rules = {}
310     for notTerminal in notTerminals:
311       rules[notTerminal] = []
312
313     for rule in lines[2:-1]:
314       rule = rule.split(" ")
315       rules[rule[0]].append(rule[2:])
316
317     return Grammar(
318       notTerminals=notTerminals,
319       terminals=terminals,
320       rules=rules,
321       start=start,
322     )
```