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) К.Э. Ковалец (И. О. Фамилия)

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

2024 г.

1 Выполнение лабораторной работы

1.1 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.

1.2 Преобразование к нормальной форме Хомского.

Определение. КС-грамматика $G = (N, \Sigma, P, S)$ называется грамматикой в (или в), 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.

1.3 Результаты работы программы

Результаты работы программы по преобразованию грамматик приведены на рисунках 1.1–1.14.

1.3.1 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 1.1 – 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' | ε
T -> F T'
T' -> * F T' | ε
F -> a | ( E )
```

Рисунок 1.2 – 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 | ε
```

Рисунок 1.3 – 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' | ε
```

Рисунок 1.4 – 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 1.5 – 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 1.6 – Грамматика после удаления левой рекурсии (пример 3)

1.3.2 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 1.7 – 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 1.8 – Грамматика после удаления левой факторизации

1.3.3 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 1.9 – 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 1.10 – 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 1.11 – 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 1.12 – 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 1.13 – 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 1.14 – Грамматика после преобразования к нормальной форме Хомского (пример 3)

2 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.

3 Текст программы

В листингах 3.1–3.3 представлен код программы.

Листинг 3.1 — Основной модуль программы

```python
import subprocess

from color import *
from grammar import Grammar, reedGrammarFromFile


MENU = f"""
    {YELLOW}\tМеню\n
    {YELLOW}1.{BASE}  Исходная грамматика;
    {YELLOW}2.{BASE}  Грамматика после устранения левой рекурсии;
    {YELLOW}3.{BASE}  Грамматика после устранения левой факторизации;
    {YELLOW}4.{BASE}  Грамматика после устранения левой рекурсии и левой
↪   факторизации;
    {YELLOW}5.{BASE}  Преобразование КС-грамматики к нормальной форме Хомского.

    {YELLOW}0.{BASE}  Выход.\n
    {GREEN}Выбор:{BASE} """


SIZE_MENU = 5
OUTPUT_FILE_NAME = "../data/result.txt"


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"{RED}\nОжидался ввод целого числа от {minOptions} до
        ↪ {maxOptions}{BASE}")

    return option


def chooseInputFile() -> str:
    with open("temp.txt", "w") as f:
```

Продолжение листинга 3.1

```
40          subprocess.run(["ls", "../data"], stdout=f)
41
42      with open("temp.txt") as f:
43          fileNames = [line[:-1] for line in f.readlines()]
44
45      subprocess.run(["rm", "temp.txt"])
46
47      msg = f"\n\t{YELLOW}Входные файлы:{BASE}\n\n"
48      for i in range(len(fileNames)):
49          msg += f"    {YELLOW}{i + 1}.{BASE}  {fileNames[i]};\n"
50      msg += f"\n    {GREEN}Выбор:{BASE} "
51
52      option = -1
53      while option == -1:
54          option = option = inputOption(
55              minOptions=1,
56              maxOptions=len(fileNames),
57              msg=msg,
58          )
59
60      return f"../data/{fileNames[option - 1]}"
61
62
63  def main():
64      inputFile = chooseInputFile()
65      option = -1
66      while option != 0:
67          option = inputOption(
68              minOptions=0,
69              maxOptions=SIZE_MENU,
70              msg=MENU,
71          )
72          match option:
73              case 1:
74                  grammar: Grammar = reedGrammarFromFile(inputFile)
75                  grammar.printGrammar()
76              case 2:
77                  grammar: Grammar = reedGrammarFromFile(inputFile)
78                  grammar.removeLeftRecursion()
79                  grammar.printGrammar()
80                  grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
81              case 3:
82                  grammar: Grammar = reedGrammarFromFile(inputFile)
83                  grammar.removeLeftFactorization()
84                  grammar.printGrammar()
```

Продолжение листинга 3.1

```
                grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
            case 4:
                grammar: Grammar = reedGrammarFromFile(inputFile)
                grammar.removeLeftRecursion()
                grammar.removeLeftFactorization()
                grammar.printGrammar()
                grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
            case 5:
                grammar: Grammar = reedGrammarFromFile(inputFile)
                grammar.convertToChomskyForm()
                grammar.printGrammar()
                grammar.createFileFromGrammar(OUTPUT_FILE_NAME)


if __name__ == '__main__':
    main()
```

Листинг 3.2 — Модуль для преобразования грамматик

```
from functools import reduce
from copy import deepcopy


class Grammar:
    notTerminals: list[str]
    terminals: list[str]
    rules: dict[str, list[list[str]]]
    start: str

    def __init__(
        self,
        notTerminals: list[str],
        terminals: list[str],
        rules: dict[str, list[list[str]]],
        start: str
    ) -> None:
        self.notTerminals = notTerminals
        self.terminals = terminals
        self.rules = rules
        self.start = start

    def printGrammar(self) -> None:
        notTerminals = Grammar.__joinListWithSymbol(self.notTerminals, ", ")
        terminals = Grammar.__joinListWithSymbol(self.terminals, ", ")

```

Продолжение листинга 3.2

```
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
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}")
```

Продолжение листинга 3.2

```
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]
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] == "Epselen":
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:
```

Продолжение листинга 3.2

```
                            self.notTerminals.append(f"{rightRule[0]}'")
                            self.rules[f"{rightRule[0]}'"] = [[rightRule[0]]]

                    rightRule = rightRule[1:]
                    newNotTerminal = f"<{"".join(rightRule)}>"
                    while len(rightRule) > 2:
                        if not newNotTerminal in self.notTerminals:
                            self.notTerminals.append(newNotTerminal)

                            if rightRule[0] in self.notTerminals:
                                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:
                        self.notTerminals.append(newNotTerminal)
                        self.rules[newNotTerminal] = [rightRule]

    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]}")
            f.write("\n")

            for notTerminal in self.notTerminals:
                for rightRule in self.rules[notTerminal]:
                    f.write(f"{notTerminal} ->")
```

Продолжение листинга 3.2

```
                    for symbol in rightRule:
                        f.write(f" {symbol}")
                    f.write("\n")

            f.write(f"{self.start}\n")

    def __removeDirectLeftFactorization(self, notTerminal: str, maxPrefix: str) ->
    ↪ None:
        indexList = self.__findPrefixMatches(
            rightRules=self.rules[notTerminal],
            prefix=maxPrefix,
        )
        newRightRules = []
        lenMaxPrefix = len(maxPrefix)
        for i in indexList:
            if len(self.rules[notTerminal][i]) > lenMaxPrefix:
                newRightRules.append(self.rules[notTerminal][i][lenMaxPrefix:])
            else:
                newRightRules.append(["Epselen"])

        rightRules = []
        for i in range(len(self.rules[notTerminal])):
            if not i in indexList:
                rightRules.append(self.rules[notTerminal][i])

        newNotTerminal = self.__findNewNotTerminal(notTerminal)
        self.rules[newNotTerminal] = newRightRules
        self.rules[notTerminal] = \
            [list(maxPrefix) + [newNotTerminal]] + rightRules

        indexNotTerminal = self.notTerminals.index(notTerminal)
        self.notTerminals = \
            self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
            self.notTerminals[indexNotTerminal + 1:]

    def __findNewNotTerminal(self, notTerminal: str):
        try:
            baseNotTerminal = notTerminal[:notTerminal.index("'")]
        except ValueError:
            baseNotTerminal = notTerminal

        quotationMarkCount = 0
        for item in self.notTerminals:
            if item.find(baseNotTerminal) != -1:
                quotationMarkCount += 1
```

Продолжение листинга 3.2

```
196
197         return notTerminal + "'" * quotationMarkCount
198
199     def __findPrefixMatches(self, rightRules: list[list[str]], prefix: str) ->
    ↪  list[int]:
200         indexList = []
201         for i in range(len(rightRules)):
202             if self.__comparePrefixes(rightRules[i], prefix):
203                 indexList.append(i)
204
205         return indexList
206
207     def __comparePrefixes(self, rightRule: list[str], prefix: str) -> bool:
208         if len(rightRule) < len(prefix):
209             return False
210
211         for i in range(len(prefix)):
212             if prefix[i] != rightRule[i]:
213                 return False
214
215         return True
216
217     def __replaceProducts(self, notTerminal: str, replaceableNotTerminal: str) ->
    ↪  None:
218         flagReplace = False
219         newRightRules = []
220         rightRules = self.rules[notTerminal]
221         for i in range(len(rightRules)):
222             if replaceableNotTerminal not in rightRules[i]:
223                 newRightRules.append(rightRules[i])
224                 continue
225
226             flagReplace = True
227             j = rightRules[i].index(replaceableNotTerminal)
228             for substitutedRightRule in self.rules[replaceableNotTerminal]:
229                 newRightRule = rightRules[i][:j]
230                 if substitutedRightRule[0] != "Epselen":
231                     newRightRule.extend(substitutedRightRule)
232                 newRightRule.extend(rightRules[i][j + 1:])
233                 newRightRules.append(newRightRule)
234
235         if flagReplace:
236             self.rules[notTerminal] = newRightRules
237             print(f"\nПосле замены {replaceableNotTerminal}: ", end="")
238             self.__printProduct(notTerminal, newRightRules)
```

Продолжение листинга 3.2

```
239
240         def __removeDirectLeftRecursion(self, notTerminal: str) -> bool:
241             self.rules[notTerminal].sort(
242                 key=lambda rightRule: rightRule[0] != notTerminal
243             )
244             newNotTerminal = notTerminal + "'"
245             rightRulesForNewNotTerminal = []
246             rightRules = []
247
248             for rightRule in deepcopy(self.rules[notTerminal]):
249                 if rightRule[0] != notTerminal:
250                     if rightRule[0] == "Epselen":
251                         rightRule = [newNotTerminal]
252                     else:
253                         rightRule.append(newNotTerminal)
254                     rightRules.append(rightRule)
255                 else:
256                     rightRule = rightRule[1:]
257                     rightRule.append(newNotTerminal)
258                     rightRulesForNewNotTerminal.append(rightRule)
259
260             if len(rightRulesForNewNotTerminal):
261                 rightRulesForNewNotTerminal.append(["Epselen"])
262                 indexNotTerminal = self.notTerminals.index(notTerminal)
263                 self.notTerminals = \
264                     self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
265                     self.notTerminals[indexNotTerminal + 1:]
266                 self.rules[newNotTerminal] = rightRulesForNewNotTerminal
267                 self.rules[notTerminal] = rightRules
268
269                 removedFlag = True
270             else:
271                 removedFlag = False
272
273             return removedFlag
274
275         def __printProduct(self, notTerminal: str, rightRules: list[list[str]]):
276             print(f"{notTerminal} -> ", end="")
277             for i in range(len(rightRules)):
278                 print(f"{" | " if i != 0 else
                    ↪  ""}{Grammar.__joinListWithSymbol(rightRules[i], " ")}", end="")
279             print()
280
281         @staticmethod
282         def __joinListWithSymbol(arr: list[str], symbol: str) -> str:
```

Продолжение листинга 3.2

```
283             return reduce(lambda elemPrev, elem: f"{elemPrev}{symbol}{elem}", arr)
284
285
286     def reedGrammarFromFile(fileName: str) -> Grammar:
287         with open(fileName) as f:
288             lines = [line[:-1] for line in f.readlines()]
289
290         notTerminals = lines[0].split(" ")
291         terminals = lines[1].split(" ")
292         start = lines[-1]
293         rules = {}
294         for notTerminal in notTerminals:
295             rules[notTerminal] = []
296
297         for rule in lines[2:-1]:
298             rule = rule.split(" ")
299             rules[rule[0]].append(rule[2:])
300
301         return Grammar(
302             notTerminals=notTerminals,
303             terminals=terminals,
304             rules=rules,
305             start=start,
306         )
```

Листинг 3.3 — Модуль с вариантами цветов при выводе сообщений в консоль

```
1  BASE   = "\x1B[0m"
2  GREEN  = "\x1B[32m"
3  RED    = "\x1B[31m"
4  YELLOW = "\x1B[33m"
5  BLUE   = "\x1B[34m"
6  PURPLE = "\x1B[35m"
```