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

Отчет
По лабораторной работе
По курсу «Конструирование компиляторов»
На тему
«Преобразования грамматик»

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

Москва, 2020

Оглавление

1 Цель и задачи работы

Цель работы: приобретение практических навыков реализации наиболее важных (но не всех) видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора.

Задачи работы:

1. Принять к сведению соглашения об обозначениях, принятые в литературе по теории формальных языков и грамматик и кратко описанные в приложении.
2. Познакомиться с основными понятиями и определениями теории формальных языков и грамматик.
3. Детально разобраться в алгоритме устранения левой рекурсии.
4. Разработать, тестировать и отладить программу устранения левой рекурсии.
5. Разработать, тестировать и отладить программу преобразования грамматики в соответствии с предложенным вариантом.

2 Листинг

2.1 main.py

```
1  from grammar import Grammar, GrammarException
2
3
4  if __name__ == '__main__':
5      __head = """ИУМ
6      7-23ГоринДмитрийВариант
7
8       3 удаление( eps правил-)
9      """
10     print(__head)
11
12     print('Тест устранения епсправил-')
13     g1 = Grammar.init_from_json_file('test_grammar_2_4_11.json')
14     print(g1)
15     print()
16     g2 = g1.delete_eps_rules()
17     print(g2)
18     print()
19     g_ans = Grammar.init_from_json_file('test_grammar_2_4_11_ans.
           json')
20     print(g_ans)
21     print()
22     assert g2 == g_ans
23
24     g1 = Grammar.init_from_json_file('test_grammar_2_23.json')
25     print(g1)
```

```
26      print()
27      g2 = g1.delete_eps_rules()
28      print(g2)
29      print()
30      g_ans = Grammar.init_from_json_file('test_grammar_2_23_ans.json'
            )
31      print(g_ans)
32      print()
33      assert g2 == g_ans
34
35      print('Тест устранения левой рекурсии:')
36      g1 = Grammar.init_from_json_file('test_grammar_4_9.json')
37      print(g1)
38      print()
39      g2 = g1.delete_left_recursion()
40      print(g2)
41      print()
42      g_ans = Grammar.init_from_json_file('test_grammar_4_9_ans.json')
43      print(g_ans)
44      print()
45      assert g2 == g_ans
46
47      print('Успешно')
```

2.2 grammar.py

```
1  import json
2  from typing import List, Dict, Tuple, Union, Iterable, Any, Set
3
4
5  class GrammarException(Exception):
6      def __init__(self, msg: str='Неизвестная ошибка грамматики', *args,
           **kwargs):
7          self.msg = msg
8          self.given_args = args
9          self.given_kwargs = kwargs
10
11     def __str__(self):
12         return f'{self.msg}, args = {self.given_args}, kwargs = {
               self.given_kwargs}'
13
14     def __repr__(self):
15         return str(self)
16
17
18 class TermSymbol:
19     """Классдлятерминальногосимвола
20
21     """
22     def __init__(self, symbol: str):
23         if symbol.lower() != symbol:
```

```
24              raise GrammarException('Символ для терминала должен быть
                    в нижнем регистре', symbol)
25          self.symbol = symbol
26
27      def __eq__(self, other):
28          if isinstance(other, str):
29              return self.symbol == other
30          elif isinstance(other, TermSymbol):
31              return self.symbol == other.symbol
32          elif isinstance(other, NoTermSymbol):
33              return False
34          else:
35              return False
36
37      def __len__(self):
38          return len(self.symbol)
39
40      def __hash__(self):
41          return hash(self.symbol)
42
43      def __str__(self):
44          return self.symbol
45
46      def __repr__(self):
47          # return fТерминал' "{self.symbol}"'
48          return f'{self.symbol}(t)'
49
50
51  class NoTermSymbol:
52      """Классдлянетерминальногосимвола
53
54      """
55      def __init__(self, symbol: str, is_initial: bool = False):
56          if symbol.upper() != symbol:
57              raise GrammarException('Символ для нетерминала должен
                    быть в верхнем регистре', symbol)
58          self.symbol = symbol
59          self.is_initial = is_initial
60
61      def __eq__(self, other):
62          if isinstance(other, str):
63              return self.symbol == other
64          elif isinstance(other, NoTermSymbol):
65              return self.symbol == other.symbol
66          elif isinstance(other, TermSymbol):
67              return False
68          else:
69              return False
70
71      def __len__(self):
```

```
72          return len(self.symbol)
73
74      def __hash__(self):
75          return hash(self.symbol)
76
77      def __str__(self):
78          return self.symbol
79
80      def __repr__(self):
81          # return fНетерминал' {self.symbol}, is_initial: {self.
              is_initial}'
82          return f'{self.symbol}(nt{"i" if self.is_initial else ""})'
83
84
85  class Grammar:
86      """Классдлягрампматики
87
88      """
89      def __init__(self, name: str,
90                   terms: Iterable[Union[str, TermSymbol]],
91                   nterms: Iterable[Union[str, NoTermSymbol]],
92                   eps_terminal: Union[str, TermSymbol],
93                   start_nterm: Union[str, NoTermSymbol],
94                   rules: Dict[Union[str, NoTermSymbol], List[List[
                       Union[str, TermSymbol, NoTermSymbol]]]]):
95          if start_nterm not in nterms:
96              raise GrammarException('Начального нетерминала нет в списке
                   нетерминалов',
97                                     start_nterm=start_nterm, nterms=
                                         nterms)
98          if eps_terminal not in terms:
99              raise GrammarException('Пустого символа нет в списке
                   терминалов',
100                                    eps_terminal=eps_terminal, terms=
                                         terms)
101         if len(eps_terminal) > 1:
102             raise GrammarException('Епсилонсимвол- должен быть длинны
                    1', eps_terminal=eps_terminal)
103         if len(nterms) < len(rules.keys()):
104             raise GrammarException('Нетерминалов больше, чем правил',
                    nterms=nterms, rules=rules)
105         self.name = name
106         self.terms = set([x if isinstance(x, TermSymbol) else
                TermSymbol(x) for x in terms])
107         self.nterms = set([x if isinstance(x, NoTermSymbol) else
                NoTermSymbol(x, x == start_nterm) for x in nterms])
108         self.eps_terminal = eps_terminal if isinstance(eps_terminal,
                TermSymbol) else TermSymbol(eps_terminal)
109         self.rules = dict()
110         for lhs, rhs in rules.items():
```

```
111             nterm = [x for x in self.nterms if x == lhs]
112             if len(nterm) == 0:
113                 raise GrammarException('Правило вывода содержит
                        нетерминал, которого нет в списке нетерминалов',
114                                        lhs=lhs, rhs=rhs, nterms=self
                                           .nterms)
115             cur_lhs = nterm[0]
116             cur_rhs = []
117             for rhs_part in rhs:
118                 if eps_terminal in rhs_part and len(rhs_part) > 1:
                        # Затирканенужных epsсимволов-
119                     rhs_part.remove(eps_terminal)
120                 cur_part = []
121                 for operand in rhs_part:
122                     if operand in self.terms:
123                         cur_part.append([x for x in self.terms if x
                                == operand][0])
124                     elif operand in self.nterms:
125                         cur_part.append([x for x in self.nterms if x
                                == operand][0])
126                     else:
127                         raise GrammarException('Правило вывода
                                содержит символ, которого нет в грамматике',
128                                                lhs=lhs, rhs=rhs,
                                                   rhs_part=rhs_part,
                                                    nterms=self.
                                                   nterms, terms=self
                                                   .terms)
129                 cur_rhs.append(cur_part)
130             self.rules[cur_lhs] = cur_rhs
131
132     @property
133     def initial_nterm(self):
134         return [x for x in self.nterms if x.is_initial][0]
135
136     @property
137     def eps_rules(self) -> List[NoTermSymbol]:
138         ans = []
139         for lhs, rhs in self.rules.items():
140             eps_rule_lst = [x for x in rhs if x == [self.
                    eps_terminal]]
141             if len(eps_rule_lst) > 0:
142                 ans.append(lhs)
143         return ans
144
145     @staticmethod
146     def init_from_json_file(filename: str):
147         """Созданиеграмматикииз
148             jsonфайла-
149         :param filename: Имяфайла
```

```
150         :return: Грамматика
151         """
152         with open(filename, 'r') as f:
153             data = json.loads(f.read())
154         try:
155             return Grammar(name=data['name'], terms=data['terms'],
                    nterms=data['nterms'],
156                            eps_terminal=data['eps_terminal'],
                               start_nterm=data['start_nterm'], rules
                               =data['rules'])
157         except KeyError:
158             raise GrammarException('Неправильный формат входного
                    файла')
159
160     def save_to_file(self, filename: str):
161         """Сохранениеграмматикивфайл
162
163         :param filename: Имяфайла
164         """
165         data = {
166             'name': self.name,
167             'terms': [x.symbol for x in self.terms],
168             'nterms': [x.symbol for x in self.nterms],
169             'eps_terminal': self.eps_terminal.symbol,
170             'start_nterm': self.initial_nterm.symbol,
171             'rules': dict()
172         }
173         for lhs, rhs in self.rules.items():
174             data['rules'][lhs.symbol] = [[y.symbol for y in x] for x
                    in self.rules[lhs]]
175         with open(filename, 'w') as f:
176             f.write(json.dumps(data, indent=4))
177
178     def pretty_string(self):
179         ans = ''
180         for lhs, rhs in self.rules.items():
181             ans += f'{lhs} -> '
182             for rule in rhs:
183                 ans += f'{"".join([x.symbol for x in rule])} | '
184             ans = ans[:-2] + '\n'
185         ans = ans[:-1]
186         return ans
187
188     def has_circuits(self, raise_exception: bool = False) -> bool:
189         """Естьлицепныеправилавграмматике Нетерминал
190
191         <> -> Нетерминал <>
192         """
193         for lhs, rhs in self.rules.items():
194             for single_rule in rhs:
```

```
195                 if len(single_rule) == 1 and isinstance(single_rule
                        [0], NoTermSymbol) and lhs != self.initial_nterm:
196                     if raise_exception:
197                         raise GrammarException('Грамматика содержит
                                цепи')
198                     return True
199         return False
200
201     # MARK: - Eps-rules removing
202
203     def find_eps_generative_nterms(self) -> List[NoTermSymbol]:
204         """Поиск
205          epsпорождающих- нетерминалов
206         :return: Списокнетерминалов
207         """
208         ans = set(self.eps_rules)
209         i = 0
210         while i < len(self.rules.items()):
211             lhs, rhs = list(self.rules.items())[i]
212             if lhs in ans:
213                 i += 1
214                 continue
215             for single_rule in rhs:
216                 single_rule_set = set(single_rule)
217                 if len(single_rule_set.difference(ans)) == 0:
218                     ans.add(lhs)
219                     i = 0
220                     break
221             else:
222                 i += 1
223         return list(ans)
224
225     def __generate_all_possibke_comb_of_objs(self, objs: List) ->
            List[List]:
226         """Рекурсивнаячастьгенерациикомбинациибез
227             ( пустого)
228         """
229         if len(objs) == 1:
230             return [[objs[0]]]
231         ans = [objs]
232         for i in range(len(objs)):
233             ans.extend(self.__generate_all_possibke_comb_of_objs(
                    objs[:i] + objs[i + 1:]))
234         return ans
235
236     def __generate_all_possible_comb_of_nterms(self, i_st, i_end,
            seq, single_rule) -> List[Union[TermSymbol, NoTermSymbol]]:
237         """Генерациявсехвозможныхкомбинацийепснетерминалов
238             - для( удаленияепсправил -)
239         """
```

```
240         all_combs_ = self.__generate_all_possibke_comb_of_objs(seq)
241         all_combs = []
242         for x in all_combs_:
243             if x not in all_combs:
244                 all_combs.append(x)
245         all_combs.append([])
246         ans = []
247         for comb in all_combs:
248             appendee = single_rule[:i_st] + comb + single_rule[i_end
                :]
249             if len(appendee) != 0:
250                 ans.append(appendee)
251         return ans
252
253     def __find_all_nterm_eps_seq(self, single_rule, eps_gen_nterms)
        -> List[Tuple[List[NoTermSymbol], int, int]]:
254         """Поисквсехпоследовательностейепснетерминалов
255            - водномвыводе
256         """
257         ans = []
258         cur_seq = []
259         i_st, i_end = 0, 0
260         for i, sym in enumerate(single_rule):
261             if sym in eps_gen_nterms:
262                 cur_seq.append(sym)
263                 i_end = i
264             else:
265                 if len(cur_seq) == 0:
266                     i_st += 1
267                     continue
268                 # i_st = 0 if i_st == 0 else i_st + 1
269                 i_end += 1  #
                    Указываетнаследующийпослепоследнеговпоследовательностиепснетерминал
                    -
270                 ans.append((cur_seq, i_st, i_end))
271                 cur_seq = []
272                 i_st = i_end+1
273         if len(cur_seq) > 0:
274             # i_st = 0 if i_st == 0 else i_st + 1
275             i_end += 1  #
                Указываетнаследующийпослепоследнеговпоследовательностиепснетерминалов
                -
276             ans.append((cur_seq, i_st, i_end))
277         return ans
278
279     def __generate_new_rhs(self, rhs, eps_gen_nterms) -> List[List[
        Union[TermSymbol, NoTermSymbol]]]:
280         """Генерацияновыхвыводовприудалениипсправил
281             -
282         """
```

```
283         ans = set()
284         already_computed = set()
285         for single_rule in rhs:
286             ans.add(tuple(single_rule))
287             queue = [single_rule]
288             while len(queue) > 0:
289                 current_rule = queue.pop()
290                 if tuple(current_rule) in already_computed:
291                     continue
292                 already_computed.add(tuple(current_rule))
293                 all_nterm_seq = self.__find_all_nterm_eps_seq(
                        current_rule, eps_gen_nterms)
294                 for nterm_seq in all_nterm_seq:
295                     to_add = self.
                            __generate_all_possible_comb_of_nterms(
                            nterm_seq[1], nterm_seq[2], nterm_seq[0],
296                                                                    curr
                                                                        )

297                     for addee in to_add:
298                         if tuple(addee) not in already_computed:
299                             queue.append(addee)
300                         ans.add(tuple(addee))
301         return [list(x) for x in ans]
302
303     def delete_eps_rules(self):
304         """Удаление
305          epsправил–
306         :return: Грамматикабезепсправил  –
307         """
308         eps_gen_nterms = self.find_eps_generative_nterms()
309         new_rules = self.rules.copy()
310         new_nterms = set([x for x in self.nterms])
311         new_initial = self.initial_nterm
312         for lhs, rhs in self.rules.items():
313             new_rules[lhs] = self.__generate_new_rhs(rhs,
                    eps_gen_nterms)
314             # Удаляемепспереход –
315             new_rules[lhs] = [x for x in new_rules[lhs] if x != [
                    self.eps_terminal]]
316         # Еслиизстартаможнополучить      eps, тозаменяемстартовоесостояние
317         if self.initial_nterm in eps_gen_nterms:
318             new_nterms.remove(self.initial_nterm)
319             new_nterms.add(NoTermSymbol(self.initial_nterm.symbol,
                    is_initial=False))
320             new_initial = NoTermSymbol(f'{self.initial_nterm.symbol
                    }\'', is_initial=True)
321             new_nterms.add(new_initial)
322             new_rules[new_initial] = [[NoTermSymbol(self.
                    initial_nterm.symbol)], [TermSymbol(self.eps_terminal
```

```
                .symbol)]]
323         return Grammar(name=f'{self.name}_без_epsправил-', terms=self
                .terms, nterms=new_nterms,
324                        eps_terminal=self.eps_terminal, start_nterm=
                           new_initial, rules=new_rules)
325
326     # MARK: - Left recursion removing
327
328     def is_direct_left_recursive_rule(self, rule: NoTermSymbol) ->
            bool:
329         """Являетсялиправилонепостредственнолеворекурсивнымначальное
330             (правилоневсчет)
331         """
332         if rule not in self.rules.keys():
333             raise GrammarException('Нет_такого_правила', rule=rule)
334         if rule == self.initial_nterm:
335             return False
336         ans = any([x[0] == rule for x in self.rules[rule]])
337         return ans
338
339     def is_direct_left_recursive(self) -> bool:
340         """Являетсялиграмматиканепосредственнолеворекурсивнойначальное
341             (правилоневсчет)
342         """
343         ans = any([self.is_direct_left_recursive_rule(x) for x in
                self.rules.keys()])
344         return ans
345
346     def delete_direct_left_recursion(self):
347         """Удалениенепосредственнойлевойрекурсии
348
349         :return: Грамматикабезнепосредственнойлевойрекурсии
350         """
351         # self.has_circuits(raise_exception=True)  #
                Проверкакналичиецепейвграмматике
352         new_nterms = self.nterms.copy()
353         new_rules = dict([(x, []) for x in self.rules.keys()])
354         for lhs, rhs in self.rules.items():
355             if not self.is_direct_left_recursive_rule(lhs):  #
                    Нерассматриваемнепосредственнолеворекурсивное
356                 new_rules[lhs] = rhs.copy()
357                 continue
358             new_rule_l = NoTermSymbol(f'{lhs.symbol}\'')
359             new_nterms.add(new_rule_l)
360             recursive_single_rules = [x for x in rhs if x[0] == lhs]
                    # Леворекурсивныеправила
361             non_recursive_single_rules = [x for x in rhs if x[0] !=
                    lhs]  # Нелеворекурсивныеправила
362             new_lhs_rules = [x + [new_rule_l] for x in
                    non_recursive_single_rules]  #
```

```
                Новыеправиладлятекущего      lhs
363             new_rule_rules = [x[1:] + [new_rule_l] for x in
                    recursive_single_rules]  # Новыеправиладля    lhs'
364             new_rule_rules.append([self.eps_terminal])
365             new_rules[new_rule_l] = new_rule_rules
366             if len(new_lhs_rules) != 0:
367                 new_rules[lhs] = new_lhs_rules
368         ans = Grammar(name=f'{self.name}без непосредственной левой
                рекурсии', terms=self.terms, nterms=new_nterms,
369                       eps_terminal=self.eps_terminal, start_nterm=
                          self.initial_nterm, rules=new_rules)
370         return ans
371
372     def delete_left_recursion(self):
373         """Устранениелевойрекурсии
374
375         :return: Грамматикабезлевойрекурсии
376         """
377         self.has_circuits(raise_exception=True)  #
                Проверкананаличиецепейвграмматике
378         lhss = list(self.rules.keys())
379         new_rules = dict([(x, []) for x in self.rules.keys()])
380         new_rules[self.initial_nterm] = self.rules[self.
                initial_nterm]
381         for i in range(1, len(lhss)):
382             for j in range(i):
383                 lhs_i = lhss[i]
384                 lhs_j = lhss[j]
385                 for single_rule in self.rules[lhs_i]:
386                     if single_rule[0] == lhs_j:
387                         appendee = [x + single_rule[1:] for x in
                            self.rules[lhs_j]]
388                         new_rules[lhs_i].extend(appendee)
389                     else:
390                         new_rules[lhs_i].append(single_rule)
391         pre_ans = Grammar(name=self.name, terms=self.terms, nterms=
                self.nterms, eps_terminal=self.eps_terminal,
392                           start_nterm=self.initial_nterm, rules=
                              new_rules)
393         ans = pre_ans.delete_direct_left_recursion()
394         ans.name = f'{self.name}без левой рекурсии'
395         return ans
396
397     # MARK: - Utils (not used)
398
399     def is_eps_rule(self, rule: NoTermSymbol) -> bool:
400         """Являетсялиправило
401             epsправилом-
402         :param rule: Леваячастьправиланетерминал    ()
403         """
```

```
404         return rule in self.eps_rules
405
406     def get_direct_left_recursive_rules(self) -> List[NoTermSymbol]:
407         """Получениенепосредственнолеворекурсивныхправил
408
409         """
410         ans = [x for x in self.rules.keys() if self.
                is_direct_left_recursive_rule(x)]
411         return ans
412
413     def term_count_for_rule(self, rule: NoTermSymbol) -> int:
414         """Количествотерминаловвправойчастиправила
415
416         :param rule: Леваячастьправиланетерминал    ()
417         """
418         return len(self.find_terms_and_nterms_in_rule(rule)[0])
419
420     def nterm_count_for_rule(self, rule: NoTermSymbol) -> int:
421         """Количестванетерминаловвправойчастиправила
422
423         :param rule: Леваячастьправиланетерминал    ()
424         """
425         return len(self.find_terms_and_nterms_in_rule(rule)[1])
426
427     def find_rules_with_term_or_nterm_in_it(self, term: Union[
            NoTermSymbol, TermSymbol]) -> List[NoTermSymbol]:
428         """Поискправил
429          , вправойчастикоторыхестьнетерминалилитерминал
430         :param term: Нетерминалилитерминал
431         """
432         ans = []
433         for lhs, rhs in self.rules.items():
434             for single_rule in rhs:
435                 if term in single_rule:
436                     ans.append(lhs)
437         return ans
438
439     def find_terms_and_nterms_in_rule(self, rule: NoTermSymbol) ->
            Tuple[List[TermSymbol], List[NoTermSymbol]]:
440         """Поисктерминальныхинетерминальныхсимволовввправойчасти
441
442         """
443         terms, nterms = [], []
444         for single_rule in self.rules[rule]:
445             terms.extend([x for x in single_rule if x in self.terms
                    ])
446             nterms.extend([x for x in single_rule if x in self.
                    nterms])
447         return terms, nterms
448
```

```
449     def find_rules_with_only_terms(self, with_empty: bool = True) ->
            List[NoTermSymbol]:
450         """Поискправилгдевправойчаститолькотерминалы
451 
452         """
453         ans = []
454         for nterm in self.rules.keys():
455             terms, nterms = self.find_terms_and_nterms_in_rule(nterm
                )
456             if len(nterms) == 0:
457                 if len(terms) == 0 or ((len(terms) == 1) and
                    with_empty):
458                     ans.append(nterm)
459         return ans
460 
461     def find_rules_with_only_nterms(self) -> List[NoTermSymbol]:
462         """Поискправилгдевправойчаститольконетерминалы
463 
464         :return:
465         """
466         ans = []
467         for nterm in self.rules.keys():
468             terms, _ = self.find_terms_and_nterms_in_rule(nterm)
469             if len(terms) == 0:
470                 ans.append(nterm)
471         return ans
472 
473     def __eq__(self, other):
474         if not isinstance(other, Grammar):
475             return False
476         if self.nterms == other.nterms and self.terms == other.terms
                and self.eps_terminal == other.eps_terminal \
477             and self.initial_nterm == other.initial_nterm and self.
                    rules.keys() == other.rules.keys():
478             for lhs in self.rules.keys():
479                 rhs = set([tuple(x) for x in self.rules[lhs]])
480                 rhss = set([tuple(x) for x in other.rules[lhs]])
481                 if rhs != rhss:
482                     return False
483             return True
484         return False
485 
486     def __str__(self):
487         return f'Грамматика "{self.name}":\n{self.pretty_string()}'
```

3 Тесты

3.1 Устранение ϵ - правил

Вход:

- $S \to ABC$
- $A \to BB|\epsilon$
- $B \to CC|a$
- $C \to AA|b$

Выход:

- $S' \to S|\epsilon$
- $S \to B|ABC|A|C|BC|AC|AB$
- $A \to B|BB$
- $B \to a|CC|C$
- $C \to A|b|AA$

Вход:

- $S \to aSbS|bSaS|\epsilon$

Выход:

- $S' \to S|\epsilon$
- $S \to ba|aSb|bSa|bSaS|abS|baS|aSbS|ab$

3.2 Устранение левой рекурсии

Вход:

- $S \to Aa|b$
- $A \to Ac|Sd|\epsilon$

Выход:

- $S \to Aa|b$
- $A \to bdA'|A'$
- $A' \to cA'|adA'|\epsilon$

4 Выводы

По результатам проведенной работы студент приобрел практические навыки в реализации наиболее важных видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора. В том числе была реализована программа, принимающая грамматику, по которой строятся грамматики без ϵ - правил и без левой рекурсии

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.