Государственное образовательное учреждение
высшего профессионального образования
“Московский государственный технический университет
имени Н.Э. Баумана”

Отчет

по лабораторной работе №2
по курсу “Конструирование компиляторов”
по теме
“Преобразование грамматик”

Студент: Адамова И.О.
Группа: ИУ7-22м
Вариант: 2
Преподаватель: Ступников А.

Москва, 2020

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 устранения бесполезных символов.

Описание работы программы

Программа получает на вход 2 файла в формате JSON. В одном из них содержится грамматика, в которой следует избавиться от левой рекурсии, во втором - грамматика, в которой следует избавиться от бесполезных символов.
Для выполнения первого задания применяется следующий алгоритм:

1) В грамматике находятся эпсилон-правила и удаляются оттуда [1]
2) При помощи известного алгоритма [2] происходит устранение правил, которые в себе содержат непосредственную левую рекурсию, и правил, которые могут через цепочку продукций привести к левой рекурсии.

Для выполнения второго задания применяется в первую очередь алгоритм устранения непорождающих символов, который на первом проходе находит нетерминалы, из которых напрямую выводится терминал, после чего к множеству продуктивных символов можно добавить те символы, в которых есть продукции состоящие только из нетерминальных и порождающих символов. Все остальные нетерминалы являются непорождающими, и правила и продукции с ними могут быть удалены.
Похожим образом работает алгоритм удаления недостижимых символов. Сначала находятся достижимые символы: это начальный нетерминал и те, которые напрямую выводятся из него. Далее находятся символы, которые выводятся из вновь полученных достижимых терминалов. Остальные символы являются недостижимыми, и все правила и продукции с ними следует удалить.

Для пустой строки, которая обычно обозначается греческой буквой “эпсилон”, требуется особое обозначение. Предлагается использовать в качестве него латинскую строчную букву - “e”, и не использовать её для обозначения других терминалов.
В результате работы программы получается 2 файла в формате JSON, в которых содержатся приведенные по алгоритмам грамматики.

Тесты

Для проверки правильности работы программы были проведены следующие тесты.
Удаление левой рекурсии
На вход получен файл, в котором содержится:

```
{
        "terminals": [
             "a",
             "b",
             "c",
             "d",
             "e"
        ],
        "nonTerminals": [
             "S",
             "A"
        ],
        "startNonTerminal": "S",

        "rules": {
             "S": [["A", "a"], ["b"]],
             "A": [["A", "c"], ["S", "d"], ["e"]]
        }

}
```

Результат работы программы:

```
{
       "startNonTerminal": "S",
       "terminals": [
              "a",
              "b",
              "c",
              "d",
              "e"
       ],
       "nonTerminals": [
              "S",
              "A",
              "A_”
       ],
       "rules": {
       "S": [
              [
              "A",
              "a"
```

```
            ],
            [
            "b"
            ],
            [
            "a"
            ]
      ],
      "A": [
            [
            "c",
            "A_"
            ],
            [
            "b",
            "d",
            "A_"
            ],
            [
            "a",
            "d",
            "A_"
            ]
      ],
      "A_": [
            [
            "c",
            "A_"
            ],
            [
            "a",
            "d",
            "A_"
            ],
            [
            "e"
            ]
      ]
      }
}
```

На вход получен файл, в котором содержится:

```
{
  "terminals": [
       "a",
       "b"
  ],
  "nonTerminals": [
       "A",
       "B",
       "C"
  ],
  "startNonTerminal": "A",

  "rules": {
```

```
        "A": [["B", "C"], ["a"]],
        "B": [["C", "A"], ["A", "b"]],
        "C": [["A", "B"], ["C", "C"], ["a"]]
    }

}
```

Результат работы программы:

```
{
        "startNonTerminal": "A",
        "terminals": [
                "a",
                "b",
                "e”
        ],
        "nonTerminals": [
                "A",
                "B",
                "C",
                "B_",
                "C_"
        ],
        "rules": {
        "A": [
                [
                "B",
                "C"
                ],
                [
                "a"
                ]
        ],
        "B": [
                [
                "C",
                "A",
                "B_"
                ],
                [
                "a",
                "b",
                "B_"
                ]
        ],
        "C": [
                [
                "a",
                "C_"
                ],
                [
                "a",
                "B",
                "C_"
                ],
                [
```

```
                "a",
                "b",
                "B_",
                "C",
                "B",
                "C_"
                ]
        ],
        "B_": [
                [
                "C",
                "b",
                "B_"
                ],
                [
                "e"
                ]
        ],
        "C_": [
                [
                "C",
                "C_"
                ],
                [
                "A",
                "B_",
                "C",
                "B",
                "C_"
                ],
                [
                "e"
                ]
        ]
        }
}
```

Удаление бесполезных символов

На вход получен файл, в котором содержится:

```
{
  "terminals": [
        "a",
        "c"
  ],
  "nonTerminals": [
        "S",
        "A",
        "C",
        "D"
  ],
  "startNonTerminal": "S",

  "rules": {
        "S": [["A"], ["c"]],
        "A": [["S", "D"], ["a"]],
```

```
        "D": [["a", "D"]],
        "C": [["D"], ["a"]]
    }

}
```

Результат работы программы:

```
{
        "startNonTerminal": "S",
        "terminals": [
                "a",
                "c"
        ],
        "nonTerminals": [
                "S",
                "A",
                "C",
                "D"
        ],
        "rules": {
        "S": [
                [
                        "A"
                ],
                [
                        "c"
                ]
        ],
        "A": [
                [
                        "a"
                ]
        ]
        }
}
```

Заключение

В результате лабораторной работы была разработана программа на языке Python, принимающая на вход файлы в формате JSON, содержащие в себе грамматики. По результатам тестирования был сделан вывод, что программа успешно удаляет из них рекурсии и бесполезные символы.

Источники

1) “ Удаление эпсилон правил из грамматики”: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A3%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_eps-%D0%BF%D1%80%D0%B0%D0%B2

%D0%B8%D0%BB_%D0%B8%D0%B7_%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B8

2) “Устранение левой рекурсии”: https://neerc.ifmo.ru/wiki/index.php?title=%D0%A3%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BB%D0%B5%D0%B2%D0%BE%D0%B9_%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D0%B8

Листинг

main.py

```
from grammar import Grammar

'''
test_delete_recursion1.json - нахождение эпсилон правил
test_delete_recursion2.json - находение эпсилон правил
test_delete_recursion3.json - удаление произвольной левой рекурсии
test_delete_recursion4.json
test_useless_symbols1.json - нахождение бесполезных терминалов (здесь D
непорождающий, C недостижимый)
test_useless_symbols2.json - нахождение бесполезных символовов (остается только
одно правило)
test_useless_symbols3.json - всё остается, как было
'''


if __name__ == "__main__":
    testfile1 = 'test_delete_recursion1.json'
    grammar1 = Grammar.read_grammar_from_file(testfile1)
    print(f'Your grammar:')
    grammar1.print_grammar()
    print(f'Deleting epsilon rules...')
    grammar1.delete_epsilon_rules()
    print(f'\nGrammar without epsilon rules:')
    grammar1.print_grammar()
    print(f'Deleting left recursion...')
    grammar1.delete_left_recursion()
    print(f'\nGrammar without left recursion:')
    grammar1.print_grammar()
    grammar1.write_grammar_to_file(testfile1)

    testfile2 = 'test_useless_symbols1.json'
    grammar2 = Grammar.read_grammar_from_file(testfile2)
    print(f'\n\n\n----------------------\nYour grammar:')
    grammar2.print_grammar()
    print(f'Deleting useless symbols...')
    grammar2.delete_useless_symbols()
    print(f'Grammar without useless symbols:')
    grammar2.print_grammar()
    grammar2.write_grammar_to_file(testfile2)
```

grammar.py

```
import json
import re

class Grammar:
    def __init__(self, startNonTerminal_, terminals_: str, nonTerminals_: str,
rules_):
        self.startNonTerminal = startNonTerminal_
        self.terminals = terminals_
        self.nonTerminals = nonTerminals_
        self.rules = rules_

    def print_grammar(self):
        print(f'nonterminals: {self.nonTerminals}\nterminals:
{self.terminals}\nstart nonterminal: {self.startNonTerminal}')
        print(f'rules: {json.dumps(self.rules,indent=4)}\n')

    def read_grammar_from_file(filename: str):
        with open(filename) as json_file:
               jsonGrammar = json.load(json_file)
               return Grammar(startNonTerminal_ = jsonGrammar['startNonTerminal'],
terminals_ = jsonGrammar['terminals'],
                     nonTerminals_ = jsonGrammar['nonTerminals'], rules_ =
jsonGrammar['rules'])

    def write_grammar_to_file(self, filename: str):
        json_result = {}
        json_result['startNonTerminal'] = self.startNonTerminal
        json_result['terminals'] = self.terminals
        json_result['nonTerminals'] = self.nonTerminals
        json_result['rules'] = self.rules
        filename = 'result_' + filename
        with open(filename, 'w') as outfile:
               json.dump(json_result, outfile, indent=4)

    def find_eps_nonterms(self):
        eps_nonterms = []
        for key, value in self.rules.items():
               for alt in value:
                     if "e" in alt:
                            eps_nonterms.append(key)

        while 1:
               new_nonterms = []
               # составляем регулярное выражение из найденных эпсилон-нетерминалов
               eps_nonterms_regex = '[' + ''.join(x for x in eps_nonterms) + ']+'
               regex = re.compile(eps_nonterms_regex)
               #print(f'epsilon-nonterms regex: {eps_nonterms_regex}')

               # в каждом правиле проверям
               for key, value in self.rules.items():
                     if not(key in eps_nonterms):
                            # каждую альтернативу: если она целиком состоит из
найденных эпсилон-нетерминалов
```

```
                                # то добавляем в множество новый нетерминал
                                for alt in value:
                                        alt_string = ''.join(x for x in alt)
                                        #print(f'string to check: {alt_string}')
                                        match_res = regex.fullmatch(alt_string)
                                        if match_res:
                                                eps_nonterms += key
                                                new_nonterms += key

                        if new_nonterms == []:
                                break

                print(f'epsilon-nonterms: {eps_nonterms}')
                return eps_nonterms

        def delete_epsilon_rules(self):
                if 'e' not in self.terminals:
                        return
                eps_nonterms = self.find_eps_nonterms()
                rules_ = self.rules
                # 1)     удаляем эпсилон-правила
                # 2)    для каждой альтернативы с эпсилон-нетерминалом добавить
                #      альтернативу без него
                for key, value in self.rules.items():
                        if ["e"] in value:
                                value.remove(["e"])
                        for alternative in value:
                                current_alt = []
                                current_rest = alternative
                                for symbol in alternative:
                                        current_alt += symbol
                                        current_rest = current_rest[1:]
                                        if symbol in eps_nonterms:
                                                new_alternative = current_alt[:-1] + current_rest
                                                if new_alternative not in rules_[key] and
new_alternative != []:
                                                        rules_[key] += [new_alternative]

                # если из начального нетерминала выводится эпсилон,
                # добавляем новое начальное правило
                #print(f'eps_nonterms: {eps_nonterms}')
                if self.startNonTerminal in eps_nonterms:
                        new_rule = []
                        new_rule.append([self.startNonTerminal])
                        new_rule.append(["e"])
                        new_start_nonterm = self.startNonTerminal + "_"
                        rules_[new_start_nonterm] = new_rule
                        self.startNonTerminal = new_start_nonterm
                self.rules = rules_

        def __handle_direct_left_recursive_rule(self, nonterm_from: str, nonterm_to:
str):
                left_rec_alt =[]
                other_alt = []
```

```
        for alt in self.rules[nonterm_from]:
                if alt[0] == nonterm_to:
                        left_rec_alt += [alt]
                else:
                        other_alt += [alt]
        if left_rec_alt == []:
                return
        new_rule_for_nonterm = []
        new_nonterm = nonterm_from + "_"
        self.nonTerminals.append(new_nonterm)
        for alt in other_alt:
                if alt == ["e"]:
                        alt = [new_nonterm]
                else:
                        alt.append(new_nonterm)
        self.rules[nonterm_from] = other_alt
        rule_for_new_nonterm = []
        for alt in left_rec_alt:
                alt = alt[1:]
                alt.append(new_nonterm)
                rule_for_new_nonterm.append(alt)
        rule_for_new_nonterm.append(["e"])
        if "e" not in self.terminals:
                self.terminals.append("e")
        self.rules[new_nonterm] = rule_for_new_nonterm



    def __handle_common_left_recursive_rule(self, nonterm_from: str, nonterm_to:
str):
        new_rule = []
        numbers_of_found_alts = []
        found_alts = []

        for alt_number in range(len(self.rules[nonterm_from])):
                if self.rules[nonterm_from][alt_number][0] == nonterm_to:
                        found_alts.append(self.rules[nonterm_from][alt_number])
                        numbers_of_found_alts.append(alt_number)
        if found_alts == []:
                return
        # удаляем найденные альтернативы
        for number in numbers_of_found_alts[::-1]:
                del self.rules[nonterm_from][number]
        insert_from = self.rules[nonterm_to]
        # заменяем их на новые
        for alt in found_alts:
                for to_insert in insert_from:
                        if to_insert == ["e"]:
                                new_alt = alt[1:]
                        else:
                                new_alt = to_insert + alt[1:]
                        self.rules[nonterm_from].append(new_alt)
```

```
    def delete_left_recursion(self):
        for i in range(0, len(self.nonTerminals)):
               for j in range(0, i+1):
                      if self.nonTerminals[i] == self.nonTerminals[j]:

self.__handle_direct_left_recursive_rule(self.nonTerminals[i],
self.nonTerminals[j])
                      else:

self.__handle_common_left_recursive_rule(self.nonTerminals[i],
self.nonTerminals[j])

    def __find_productive_symbols(self):
        productive_symbols = []
        terminals_regex = '[' + ''.join(x for x in self.terminals) + ']+'
        #print(terminals_regex)
        regex = re.compile(terminals_regex)

        for key, value in self.rules.items():
               for alternative in value:
                      alt_string = ''.join(x for x in alternative)
                      isProductive = regex.fullmatch(alt_string)
                      if isProductive:
                             productive_symbols.append(key)

        while 1:
               new_symbols = []
               for key, value in self.rules.items():
                      if key not in productive_symbols:
                             for alternative in value:
                                    isProductive = True
                                    for symbol in alternative:
                                           if symbol not in productive_symbols and
symbol not in self.terminals:
                                                  isProductive = False
                                    if isProductive:
                                           if key not in productive_symbols:
                                                  productive_symbols.append(key)
                                                  new_symbols.append(key)
               if new_symbols == []:
                      break

        return productive_symbols

    def __delete_useless_symbols(self, useless_symbols):
        for symbol in useless_symbols:
               self.rules.pop(symbol, None)
        for key, value in self.rules.items():
               for alt in value:
                      if set(alt) & set(useless_symbols):
                             self.rules[key].remove(alt)
        #print(self.rules)

    def __find_reachable_symbols(self):
```

```
        reachable_symbols = [self.startNonTerminal]

        for alternative in self.rules[self.startNonTerminal]:
               for symbol in alternative:
                      if symbol in self.nonTerminals:
                             reachable_symbols.append(symbol)
        #print(f'first reachable symbols: {reachable_symbols}')

        while 1:
               new_symbols = []
               for key, value in self.rules.items():
                      if key in reachable_symbols:
                             for alternative in value:
                                    for symbol in alternative:
                                           if symbol in self.nonTerminals and symbol
not in reachable_symbols:
                                                  new_symbols.append(symbol)
               if new_symbols == []:
                      break
               reachable_symbols += new_symbols

        return reachable_symbols



    def delete_useless_symbols(self):
        productive_symbols = self.__find_productive_symbols()
        nonproductive_symbols = []
        for x in self.nonTerminals:
               if x not in productive_symbols:
                      nonproductive_symbols += x
        print(f'\nproductive_symbols: {productive_symbols}\nnonproductive_symbols:
{nonproductive_symbols}')

        self.__delete_useless_symbols(nonproductive_symbols)

        reachable_symbols = self.__find_reachable_symbols()
        nonreachable_symbols = []
        for x in self.nonTerminals:
               if x not in reachable_symbols:
                      nonreachable_symbols += x
        print(f'\nreachable_symbols: {reachable_symbols}\nnonreachable_symbols:
{nonreachable_symbols}\n')
        self.__delete_useless_symbols(nonreachable_symbols)
```