|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_\_

**ОТЧЕТ**

**по Лабораторной работе №2**

**по курсу**

**«Проектирование компиляторов»**

**Тема**

**«Преобразование грамматик»**

**Вариант 3**

Студент \_\_\_ИУ7-21М\_\_\_\_ \_\_\_\_\_Карпухин А.С.\_\_\_\_

(Группа) (И.О.Фамилия)

Преподаватель \_\_\_\_Ступников А.А.\_\_\_\_

(И.О.Фамилия)

*2021 г.*

**Задание**

**Цель работы**: приобретение практических навыков реализации наиболее важных 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 ε-правил.

**Листинги кода**

Листинг 1. Удаление левой рекурсии.

|  |
| --- |
| #include "..\..\include\grammar\_transform\left\_recursion\_remove.h"  LeftRecursionRemove::LeftRecursionRemove(Grammar& grammar)  : GrammarTransform{ grammar }  {  }  void LeftRecursionRemove::operator()()  {  symbol\_vector symbols;  for (auto i = \_grammar.nonTerminals.begin(); i != \_grammar.nonTerminals.end(); i++)  {  \_unfoldRules(i);  if (auto symbol = \_removeDirectRecursion(i))  {  symbols.push\_back(symbol);  }  }  \_grammar.nonTerminals.insert(\_grammar.nonTerminals.end(), symbols.begin(), symbols.end());  }  void LeftRecursionRemove::\_unfoldRules(const symbol\_iterator& i)  {  for (auto j = \_grammar.nonTerminals.begin(); j != i; j++)  {  rule\_vector ijRules;  rule\_vector jRules;  \_findIjRules(i, j, ijRules);  \_findIndexedRules(j, jRules);  std::erase\_if(\_grammar.rules, [&](auto r) {  return std::find(ijRules.begin(), ijRules.end(), r) != ijRules.end(); });  for (auto& ijRule : ijRules)  {  for (auto& jRule : jRules)  {  symbol\_vector right{ jRule->getRight() };  right.insert(right.end(), ++(ijRule->getRight().begin()), ijRule->getRight().end());  \_grammar.rules.push\_back(rule\_ptr{ new Rule{ ijRule->getLeft(), right } });  }  }  }  }  symbol\_ptr LeftRecursionRemove::\_removeDirectRecursion(const symbol\_iterator& i)  {  rule\_vector recRules;  rule\_vector indRules;  // Searching all Ai-rules  \_findRecursiveRules(i, recRules);  if (recRules.empty()) return nullptr;  \_findIndexedRules(i, indRules);  // Removing rules of type Ai -> Ai a  std::erase\_if(\_grammar.rules, [&](auto r) {  return std::find(recRules.begin(), recRules.end(), r) != recRules.end(); });  // Add new nonterminal symbol Аi'  symbol\_ptr symbol{ new Symbol{ (\*i)->getName() + "'", (\*i)->getSpell(), (\*i)->getType()} };  // Add rules of type Ai -> b Ai'  for (auto& rule : indRules)  {  symbol\_vector right{ rule->getRight() };  right.push\_back(symbol);  \_grammar.rules.push\_back(rule\_ptr{ new Rule{ rule->getLeft(), right } });  }  // Add rules of type Ai' -> a | a Ai'  for (auto& rule : recRules)  {  symbol\_vector right{};  right.insert(right.begin(), ++(rule->getRight().begin()), rule->getRight().end());  \_grammar.rules.push\_back(rule\_ptr{ new Rule{ { symbol }, right } });  right.push\_back(symbol);  \_grammar.rules.push\_back(rule\_ptr{ new Rule{ { symbol }, right } });  }  return symbol;  }  void LeftRecursionRemove::\_findRecursiveRules(const symbol\_iterator& it, rule\_vector& rules)  {  std::copy\_if(\_grammar.rules.begin(), \_grammar.rules.end(),  std::back\_inserter(rules), [&](auto rule)  {  return (rule->getRight().empty())  ? false  : rule->getLeft()[0] == \*it && rule->getRight()[0] == \*it;  });  }  void LeftRecursionRemove::\_findIndexedRules(const symbol\_iterator& it, rule\_vector& rules)  {  std::copy\_if(\_grammar.rules.begin(), \_grammar.rules.end(), std::back\_inserter(rules),  [&](auto rule)  {  if (rule->getRight().empty() || rule->getLeft()[0] != \*it) return false;  auto s = std::find(\_grammar.nonTerminals.begin(),  \_grammar.nonTerminals.end(), rule->getRight()[0]);  return s == \_grammar.nonTerminals.end() || s > it;  });  }  void LeftRecursionRemove::\_findIjRules(const symbol\_iterator& i, const symbol\_iterator& j, rule\_vector& rules)  {  std::copy\_if(\_grammar.rules.begin(), \_grammar.rules.end(), std::back\_inserter(rules),  [&](auto rule)  {  return (rule->getRight().empty())  ? false  : rule->getLeft()[0] == \*i && rule->getRight()[0] == \*j;  });  } |

Листинг 2. Удаление ε-правил.

|  |
| --- |
| #include "..\..\include\grammar\_transform\epsilon\_rule\_remove.h"  EpsilonRuleRemove::EpsilonRuleRemove(Grammar& grammar)  : GrammarTransform{ grammar }  {  }  void EpsilonRuleRemove::operator()()  {  symbol\_set ne{};  \_buildNeSet(ne);  \_replaceEpsilonRules(ne);  }  void EpsilonRuleRemove::\_buildNeSet(symbol\_set& ne)  {  size\_t prev{ 0 };  ne.clear();  do  {  prev = ne.size();  rule\_vector rules;  // Find all the rules whose right side is empty or consists of symbols from prev  std::copy\_if(\_grammar.rules.begin(), \_grammar.rules.end(),  std::back\_inserter(rules), [&](auto r)  {  return r->getRight().empty() || std::all\_of(  r->getRight().begin(), r->getRight().end(), [&](auto s)  {  return ne.contains(s);  });  });  for (auto& rule : rules)  {  ne.insert(rule->getLeft()[0]);  }  }  while (prev != ne.size());  }  void EpsilonRuleRemove::\_replaceEpsilonRules(const symbol\_set& symbols)  {  rule\_vector rules{};  size\_t size = \_grammar.rules.size();  for (size\_t i = 0; i < size; i++)  {  const symbol\_vector& right = \_grammar.rules[i]->getRight();  index\_vector occurrences{};  if (!right.empty())  {  rules.push\_back(\_grammar.rules[i]);  }  \_findAllOccurrences(symbols, right, occurrences);  if (occurrences.empty())  {  continue;  }  index\_combination\_vector combinations{};  \_findAllCombinations(occurrences, combinations);  for (auto& combination : combinations)  {  symbol\_vector newRight{ };  for (size\_t i = 0; i < right.size(); i++)  {  if (std::find(combination.begin(), combination.end(), i) == combination.end())  {  newRight.push\_back(right[i]);  }  }  if (!newRight.empty())  {  rule\_ptr rule{ new Rule{ \_grammar.rules[i]->getLeft(), newRight } };  if (std::none\_of(rules.begin(), rules.end(), [&](auto r) { return \*r == \*rule; }))  {  rules.push\_back(rule);  }  }  }  }  // If axiom contains in epsilon-reachable symbol set, add new axiom  if (symbols.contains(\_grammar.axiom))  {  symbol\_ptr symbol{ new Symbol{  \_grammar.axiom->getName() + "'",  \_grammar.axiom->getSpell(),  \_grammar.axiom->getType() } };  rule\_ptr rule1{ new Rule{ { symbol }, {} } };  rule\_ptr rule2{ new Rule{ { symbol }, { \_grammar.axiom } } };  \_grammar.nonTerminals.push\_back(symbol);  \_grammar.axiom = symbol;  rules.push\_back(rule1);  rules.push\_back(rule2);  }  \_grammar.rules = rules;  }  void EpsilonRuleRemove::\_findAllOccurrences(  const symbol\_set& symbols,  const symbol\_vector& right,  index\_vector& occurrences)  {  if (std::any\_of(symbols.begin(), symbols.end(), [&](auto s) { return !s; }))  {  throw std::invalid\_argument{ "Null symbol pointer in symbol vector." };  }  if (std::any\_of(right.begin(), right.end(), [&](auto s) { return !s; }))  {  throw std::invalid\_argument{ "Null pointer in right part vector." };  }  occurrences.clear();  for (auto& symbol : symbols)  {  auto it = right.begin();  while (right.end() != (it = std::find(it, right.end(), symbol)))  {  occurrences.push\_back(it - right.begin());  it++;  }  }  }  void EpsilonRuleRemove::\_findAllCombinations(  const index\_vector& indexes,  index\_combination\_vector& combinations)  {  int n = indexes.size();  for (int r = 1; r <= n; r++)  {  std::vector<bool> v(n);  std::fill(v.begin(), v.begin() + r, true);  do  {  index\_vector combination{};  for (int i = 0; i < n; i++)  {  if (v[i]) combination.push\_back(indexes[i]);  }  combinations.push\_back(combination);  } while (std::prev\_permutation(v.begin(), v.end()));  }  } |

**Пример работы программы**

1. **Пример удаления ε-правил**

Исходная грамматика:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <grammar name="G0">  <terminalsymbols>  <term name="a" spell="a" />  <term name="b" spell="b" />  </terminalsymbols>  <nonterminalsymbols>  <nonterm name="S" />  </nonterminalsymbols>  <productions>  <production>  <lhs name="S" />  <rhs>  <symbol type="term" name="a" />  <symbol type="nonterm" name="S" />  <symbol type="term" name="b" />  <symbol type="nonterm" name="S" />  </rhs>  </production>  <production>  <lhs name="S" />  <rhs>  <symbol type="nonterm" name="S" />  <symbol type="term" name="a" />  <symbol type="term" name="b" />  <symbol type="nonterm" name="S" />  </rhs>  </production>  <production>  <lhs name="S" />  <rhs>  </rhs>  </production>  </productions>  <startsymbol name="S" />  </grammar> |

Полученная грамматика без ε-правил:

|  |
| --- |
| <grammar name="G0">  <terminalsymbols>  <term name="a" spell="a"/>  <term name="b" spell="b"/>  </terminalsymbols>  <nonterminalsymbols>  <nonterm name="S"/>  <nonterm name="S’"/>  </nonterminalsymbols>  <productions>  <production>  <lhs name="S"/>  <rhs>  <symbol type="term" name="a"/>  <symbol type="nonterm" name="S"/>  <symbol type="term" name="b"/>  <symbol type="nonterm" name="S"/>  </rhs>  </production>  <production>  <lhs name="S"/>  <rhs>  <symbol type="term" name="a"/>  <symbol type="term" name="b"/>  <symbol type="nonterm" name="S"/>  </rhs>  </production>  <production>  <lhs name="S"/>  <rhs>  <symbol type="term" name="a"/>  <symbol type="nonterm" name="S"/>  <symbol type="term" name="b"/>  </rhs>  </production>  <production>  <lhs name="S"/>  <rhs>  <symbol type="term" name="a"/>  <symbol type="term" name="b"/>  </rhs>  </production>  <production>  <lhs name="S"/>  <rhs>  <symbol type="nonterm" name="S"/>  <symbol type="term" name="a"/>  <symbol type="term" name="b"/>  <symbol type="nonterm" name="S"/>  </rhs>  </production>  <production>  <lhs name="S"/>  <rhs>  <symbol type="nonterm" name="S"/>  <symbol type="term" name="a"/>  <symbol type="term" name="b"/>  </rhs>  </production>  <production>  <lhs name="S’"/>  <rhs/>  </production>  <production>  <lhs name="S’"/>  <rhs>  <symbol type="nonterm" name="S"/>  </rhs>  </production>  </productions>  <startsymbol name="S’"/>  </grammar> |

1. **Пример удаления левой рекурсии**

Исходная грамматика:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <grammar name="G0">  <terminalsymbols>  <term name="IDENT" spell="a" />  <term name="ADD" spell="+" />  <term name="MUL" spell="\*" />  <term name="LPAREN" spell="(" />  <term name="RPAREN" spell=")" />  </terminalsymbols>  <nonterminalsymbols>  <nonterm name="E" />  <nonterm name="T" />  <nonterm name="F" />  </nonterminalsymbols>  <productions>  <production>  <lhs name="E" />  <rhs>  <symbol type="nonterm" name="E" />  <symbol type="term" name="ADD" />  <symbol type="nonterm" name="T" />  </rhs>  </production>  <production>  <lhs name="E" />  <rhs>  <symbol type="nonterm" name="T" />  </rhs>  </production>  <production>  <lhs name="T" />  <rhs>  <symbol type="nonterm" name="T" />  <symbol type="term" name="MUL" />  <symbol type="nonterm" name="F" />  </rhs>  </production>  <production>  <lhs name="T" />  <rhs>  <symbol type="nonterm" name="F" />  </rhs>  </production>  <production>  <lhs name="F" />  <rhs>  <symbol type="term" name="IDENT" />  </rhs>  </production>  <production>  <lhs name="F" />  <rhs>  <symbol type="term" name="LPAREN" />  <symbol type="nonterm" name="E" />  <symbol type="term" name="RPAREN" />  </rhs>  </production>  </productions>  <startsymbol name="E" />  </grammar> |

Полученная грамматика без левой рекурсии:

|  |
| --- |
| <grammar name="G0">  <terminalsymbols>  <term name="IDENT" spell="a"/>  <term name="ADD" spell="+"/>  <term name="MUL" spell="\*"/>  <term name="LPAREN" spell="("/>  <term name="RPAREN" spell=")"/>  </terminalsymbols>  <nonterminalsymbols>  <nonterm name="E"/>  <nonterm name="T"/>  <nonterm name="F"/>  <nonterm name="E&apos;"/>  <nonterm name="T&apos;"/>  </nonterminalsymbols>  <productions>  <production>  <lhs name="E"/>  <rhs>  <symbol type="nonterm" name="T"/>  </rhs>  </production>  <production>  <lhs name="T"/>  <rhs>  <symbol type="nonterm" name="F"/>  </rhs>  </production>  <production>  <lhs name="F"/>  <rhs>  <symbol type="term" name="IDENT"/>  </rhs>  </production>  <production>  <lhs name="F"/>  <rhs>  <symbol type="term" name="LPAREN"/>  <symbol type="nonterm" name="E"/>  <symbol type="term" name="RPAREN"/>  </rhs>  </production>  <production>  <lhs name="E"/>  <rhs>  <symbol type="nonterm" name="T"/>  <symbol type="nonterm" name="E&apos;"/>  </rhs>  </production>  <production>  <lhs name="E&apos;"/>  <rhs>  <symbol type="term" name="ADD"/>  <symbol type="nonterm" name="T"/>  </rhs>  </production>  <production>  <lhs name="E&apos;"/>  <rhs>  <symbol type="term" name="ADD"/>  <symbol type="nonterm" name="T"/>  <symbol type="nonterm" name="E&apos;"/>  </rhs>  </production>  <production>  <lhs name="T"/>  <rhs>  <symbol type="nonterm" name="F"/>  <symbol type="nonterm" name="T&apos;"/>  </rhs>  </production>  <production>  <lhs name="T&apos;"/>  <rhs>  <symbol type="term" name="MUL"/>  <symbol type="nonterm" name="F"/>  </rhs>  </production>  <production>  <lhs name="T&apos;"/>  <rhs>  <symbol type="term" name="MUL"/>  <symbol type="nonterm" name="F"/>  <symbol type="nonterm" name="T&apos;"/>  </rhs>  </production>  </productions>  <startsymbol name="E"/>  </grammar> |

**Заключение**

В результате выполнения лабораторной работы были изучены алгоритмы удаления левой рекурсии и ε-правил, а также реализована программа, позволяющая выполнять следующие преобразования над грамматикой, загружаемой из файла в XML-формате:

* Построение новой грамматики, эквивалентной исходной, без ε-правил;
* Построение новой грамматики, эквивалентной исходной, без левой рекурсии.