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»

Лабораторная работа № 2

Дисциплина Конструирование компиляторов

Тема Преобразования грамматик

Вариант №02

Студент Кибамба Ж.Ж.

Группа ИУ7И-11М

Преподаватель Ступников А.А.

04.05.2024

Москва, 2023 г.

Описание задания

Устранение левой рекурсии.

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

Указания.

1. Проработать самостоятельно п 4.3.2. и п. 4.3.4. [2].
2. Воспользоваться алгоритмом 2.13. При тестировании воспользоваться примерами 4.7, 4.9 и 4.11 [2].
3. Устранять надо не только непосредственную (intermediate), но и косвенную (indirect) рекурсию. Этот вопрос подробно затронут в [4].
4. После устранения левой рекурсии можно применить левую факторизацию.

Варианта №2 – Устранение бесполезных символов.

Постройте программу, которая в качестве входа принимает произвольную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику $G' = (N', \Sigma', P', S')$, не содержащую бесполезных символов.

Указания. Воспользоваться алгоритмом 2.9. [1]. При тестировании воспользоваться примером 2.22. и упражнением 2.4.6. [1].

Текст программы

Листинг 1 – Класс устранения левой рекурсии и левой факторизации.

```
package ru.bmstu.kibamba.grammars;

import java.util.*;
import java.util.stream.Collectors;

import static ru.bmstu.kibamba.grammars.GrammarUtils.*;

public class LeftRecursionEliminator {
    private static final List<String> nonterminals = new ArrayList<>();
    private static final String[] POTENTIALS_NONTERMINALS = {"A", "B", "C", "D", "E",
"F", "G", "H", "I", "J", "K",
            "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y",
"Z"};

    private static Map<Integer, Production> createProductionMap(List<String> produc-
tionsStr) {
        Map<Integer, Production> result = new HashMap<>();
        var count = 0;
        for (String s : productionsStr) {
            String[] splitProduction = s.replaceAll("\\s", "").split("->");
            var nonterminal = splitProduction[0];
            if (!nonterminals.contains(nonterminal)) {
                nonterminals.add(nonterminal);
            }
            result.put(++count, new Production(nonterminal, splitProduction[1]));
        }
        return result;
    }

    private static int getProductionNonterminalIndex(String nonterminal) {
```

```
        for (var i = 0; i < nonterminals.size(); i++) {
            if (Objects.equals(nonterminals.get(i), nonterminal)) {
                return i;
            }
        }
        return -1;
    }

    private static boolean isProductionContainsLeftRecursion(Production production) {
        String productionNonterminal = production.getNonterminal();
        int productionNonterminalIndex = getProductionNonterminalIndex(productionNon-
terminal);

        String[] productionChains = getProductionChainsArray(production);

        for (String chain : productionChains) {
            String currentCh = chain.substring(0, 1);
            if (currentCh.equals(productionNonterminal)) {
                return true;
            }
            int currentChainFirstIndex = getProductionNonterminalIndex(currentCh);
            if (currentChainFirstIndex >= 0 && currentChainFirstIndex <= production-
NonterminalIndex) {
                return true;
            }
        }
        return false;
    }

    private static String getProductionChainStr(List<String> chains, String nontermi-
nal, boolean useSecondAlgorithm,
                                                boolean addEpsilon) {
        StringBuilder secondPart = new StringBuilder();
        StringBuilder firstPart = new StringBuilder();
        for (String chain : chains) {
            if (useSecondAlgorithm) {
                firstPart.append(chain).append("|");
            }
            secondPart.append(chain).append(nonterminal).append("'").append("|");
        }
        nonterminals.add(nonterminal.concat("'"));
        var sbStr = firstPart.append(secondPart).toString();
        if (!useSecondAlgorithm && addEpsilon) {
            sbStr = sbStr.concat("£|");
        }

        return removeChainLastOrCharacter(sbStr);
    }

    private static String getBetasAlphaProductionChainStr(List<String> betas, String
alpha) {
        StringBuilder sb = new StringBuilder();
        for (String beta : betas) {
            sb.append(beta).append(alpha).append("|");
        }
        var sbStr = sb.toString();
        return removeChainLastOrCharacter(sbStr);
    }

    private static String getBetasProductionStr(Set<String> betas) {
        StringBuilder sb = new StringBuilder();
        for (String beta : betas) {
            sb.append(beta).append("|");
        }
        return removeChainLastOrCharacter(sb.toString());
```

```
    }

    private static Production getProductionByChain(List<Production> productions, Pro-
duction production) {
        var productionsToCheck = productions.stream().
                filter(p -> p.getChain().length() == produc-
tion.getChain().length()).collect(Collectors.toList());
        for (Production pr : productionsToCheck) {
            if (areChainsEquals(pr.getChain(), production.getChain())) {
                return pr;
            }
        }
        return production;
    }

    private static boolean areChainsEquals(String firstChain, String secondChain) {
        var firstChainArray = getProductionChainsArray(firstChain);
        var secondChainArray = getProductionChainsArray(secondChain);
        var count = firstChainArray.length;

        for (String currentChain : firstChainArray) {
            for (String chain : secondChainArray) {
                if (currentChain.equals(chain)) {
                    count--;
                    break;
                }
            }
        }
        return count == 0;
    }

    private static void performsStep01(Map<Integer, Production> productionMap, bool-
ean useSecondAlgorithm) {
        var i = 1;
        var n = productionMap.size();
        performsStep02(n, i, productionMap, useSecondAlgorithm);
    }

    private static void performsStep02(int n, int i, Map<Integer, Production> produc-
tionMap,
                                       boolean useSecondAlgorithm) {
        var currentProduction = productionMap.get(i);
        if (isProductionContainsLeftRecursion(currentProduction)) {
            String currentProductionNonterminal = currentProduction.getNonterminal();

            List<String> alpha = new ArrayList<>();
            List<String> beta = new ArrayList<>();

            String[] productionChains = getProductionChainsArray(currentProduction);

            for (String chain : productionChains) {
                if (chain.substring(0, 1).equals(currentProductionNonterminal)) {
                    alpha.add(chain.substring(1));
                } else {
                    beta.add(chain);
                }
            }
            var currentProductionModifiedChain = getProductionChainStr(beta, cur-
rentProductionNonterminal,
                    useSecondAlgorithm, false);
            currentProduction.setChain(currentProductionModifiedChain);
            var newNonterminalProduction = currentProductionNonterminal.concat("'");
            var newNonterminalChain = getProductionChainStr(alpha, currentProduction-
Nonterminal,
                    useSecondAlgorithm, true);
```

```
            var newProduction = new Production(newNonterminalProduction, newNontermi-
nalChain);
            productionMap.put(productionMap.size() + 1, newProduction);
        }
        performsStep03(n, i, productionMap, useSecondAlgorithm);
    }

    private static void performsStep03(int n, int i, Map<Integer, Production> produc-
tionMap,
                                       boolean useSecondAlgorithm) {
        if (i == n) {
            return;
        }
        i++;
        var j = 1;
        performsStep04(n, i, j, productionMap, useSecondAlgorithm);
    }

    private static void performsStep04(int n, int i, int j, Map<Integer, Production>
productionsMap,
                                       boolean useSecondAlgorithm) {
        var ai = productionsMap.get(i);
        var result = new Production(ai.getNonterminal());
        var aiChains = getProductionChainsArray(ai);

        for (String chain : aiChains) {
            int firstNonterminalIndex = getProductionNonterminalIndex(chain.sub-
string(0, 1));
            var resultChain = new StringBuilder();
            if (firstNonterminalIndex >= 0 && firstNonterminalIndex + 1 == j) {
                List<String> betas = new ArrayList<>();
                var alpha = chain.substring(1);
                var aj = productionsMap.get(j);
                var ajChains = getProductionChainsArray(aj);
                Collections.addAll(betas, ajChains);
                resultChain.append(getBetasAlphaProductionChainStr(betas, alpha));
            } else {
                resultChain.append(chain);
            }

            result.setChain(result.getChain()
                    .concat(resultChain.toString())
                    .concat("|"));
        }
        result.setChain(removeChainLastOrCharacter(result.getChain()));
        ai.setChain(result.getChain());

        performsStep05(n, i, j, productionsMap, useSecondAlgorithm);
    }

    private static void performsStep05(int n, int i, int j, Map<Integer, Production>
productionMap,
                                       boolean useSecondAlgorithm) {
        if (j == i - 1) {
            performsStep02(n, i, productionMap, useSecondAlgorithm);
        } else {
            j++;
            performsStep04(n, i, j, productionMap, useSecondAlgorithm);
        }
    }

    public static Grammar removeLeftRecursion(Grammar grammar, boolean useSecondAlgo-
rithm) {
        Grammar clonedGrammar = grammar.clone();
```

```
        Map<Integer, Production> productionMap = createProductionMap(getProduc-
tionsStr(clonedGrammar
                .getProductions()));
        performsStep01(productionMap, useSecondAlgorithm);
        List<Production> productions = buildProduction(productionMap);
        productions = eliminateEpsilonFactor(productions);

        return new Grammar(getNonterminals(productions),
                getTerminals(productions),
                productions,
                clonedGrammar.getFirstSymbol());
    }

    public static Grammar leftFactorsProduction(Grammar grammarWithoutLeftRecursion)
{
        List<Production> productions = cloneProductions(grammarWithoutLeftRecur-
sion.getProductions());
        var index = getProductionToLeftFactoriseIndex(productions);
        while (index != -1) {
            leftFactorsProduction(productions.get(index), productions);
            index = getProductionToLeftFactoriseIndex(productions);
        }

        productions = eliminateEpsilonFactor(productions);
        return new Grammar(getNonterminals(productions),
                getTerminals(productions),
                productions,
                grammarWithoutLeftRecursion.getFirstSymbol());
    }

    private static int getProductionToLeftFactoriseIndex(List<Production> produc-
tions) {
        for (var i = 0; i < productions.size(); i++) {
            var alpha = findMaxChainFactor(getProductionChainsArray(produc-
tions.get(i)));
            if (!alpha.isEmpty()) {
                return i;
            }
        }
        return -1;
    }

    private static String findMaxChainFactor(String[] chains) {
        List<String> sortedChains = Arrays.stream(chains).sorted(Comparator.compar-
ingInt(String::length).reversed())
                .collect(Collectors.toList());

        for (String currentChain : sortedChains) {
            var currentChainLength = currentChain.length();
            var chainsToCheck = Arrays.stream(chains).filter(chain -> chain.length()
>= currentChainLength)
                    .collect(Collectors.toList());
            var count = 0;
            for (String chain : chainsToCheck) {
                if (chain.startsWith(currentChain)) {
                    count++;
                }
            }
            if (count >= 2) {
                return currentChain;
            }
        }
        return findChainFactor(chains);
    }
```

```
    private static String first(String chain) {
        if (chain.length() >= 2) {
            if (chain.substring(0, 2).contains("'")) {
                var i = 2;
                var keepSearching = chain.substring(i).startsWith("'");
                while (keepSearching) {
                    i++;
                    keepSearching = chain.substring(i).startsWith("'");
                }
                return chain.substring(0, i);
            }
        }
        return chain.substring(0, 1);
    }

    private static String findChainFactor(String[] chains) {
        var i = 0;
        var count = 0;
        var first = "";
        do {
            first = first(chains[i]);
            count = count(chains, first);
            i++;
        } while (count <= 1 && i < chains.length);
        return count >= 2 ? first : "";

    }

    private static int count(String[] chains, String first) {
        var count = 0;
        for (String chain : chains) {
            if (chain.startsWith(first)) {
                count++;
            }
        }
        return count;
    }

    private static void leftFactorsProduction(Production production, List<Production>
productions) {
        var nonterminal = production.getNonterminal();
        var chains = getProductionChainsArray(production);
        var alpha = findMaxChainFactor(chains);
        Set<String> betas = new HashSet<>();

        var factor = getFactor(nonterminal);
        var firstAlphaIndex = createBetaListReturnFirstAlphaIndex(chains, alpha, be-
tas);

        Production productionToAdd = new Production(factor, getBetasProductionStr(be-
tas));
        Production productionByChain = getProductionByChain(productions, produc-
tionToAdd);
        boolean canAddProduction = factor.equals(productionByChain.getNonterminal());
        factor = productionByChain.getNonterminal();

        String modifiedChain = modifyChain(alpha, chains, firstAlphaIndex, factor);
        modifyProduction(modifiedChain, production, canAddProduction, produc-
tionToAdd, productions);
    }

    private static String modifyChain(String alpha, String[] chains, int
firstAlphaIndex, String factor) {
        StringBuilder modifiedChain = new StringBuilder();
        if (!alpha.isEmpty()) {
```

```
            for (var i = 0; i < chains.length; i++) {
                if (i == firstAlphaIndex) {
                    modifiedChain.append(alpha).append(factor).append("|");
                } else {
                    if (!chains[i].startsWith(alpha)) {
                        modifiedChain.append(chains[i]).append("|");
                    }
                }
            }
            if (!nonterminals.contains(factor)) {
                nonterminals.add(factor);
            }
        }

        return modifiedChain.toString();
    }

    private static void modifyProduction(String modifiedChain, Production production,
boolean canAddProduction,
                                         Production productionToAdd, List<Production>
productions) {

        if (!modifiedChain.isEmpty()) {
            var result = removeChainLastOrCharacter(modifiedChain);
            production.setChain(result);
            if (canAddProduction) {
                productions.add(productionToAdd);
            }
            leftFactorsProduction(production, productions);
        }
    }

    private static int createBetaListReturnFirstAlphaIndex(String[] chains, String
alpha, Set<String> betas) {
        var firstAlphaIndex = 0;
        var firstAlphaIndexHasBeenFound = false;
        var count = 0;
        for (String chain : chains) {
            if (chain.startsWith(alpha)) {
                if (!firstAlphaIndexHasBeenFound) {
                    firstAlphaIndexHasBeenFound = true;
                    firstAlphaIndex = count;
                }
                var beta = chain.substring(alpha.length());
                beta = beta.isEmpty() ? "£" : beta;
                betas.add(beta);
            }
            count++;
        }
        return firstAlphaIndex;
    }

    private static String getFactor(String nonterminal) {
        StringBuilder sb = new StringBuilder(nonterminal.contains("'") ? nonterminal
: nonterminal.concat("'"));
        if (nonterminals.contains(sb.toString())) {
            var i = 0;
            var j = 0;
            var quotationMark = getQuotationsMark(j);
            var newNonterminal = new StringBuilder(POTENTIALS_NONTERMINALS[i].con-
cat(quotationMark));
            do {
                i++;
                if (i == POTENTIALS_NONTERMINALS.length) {
                    i = 0;
```

```
                    j++;
                    quotationMark = getQuotationsMark(j);
                }
                newNonterminal = new StringBuilder(POTENTIALS_NONTERMINALS[i].con-
cat(quotationMark));
            } while (nonterminals.contains(newNonterminal.toString()));

            return newNonterminal.toString();
        }
        return sb.toString();
    }

    private static String getQuotationsMark(int i) {
        return "'".repeat(Math.max(0, i - 1));
    }
}
```

Листинг 2 – Класс проверки пустоты языка

```
package ru.bmstu.kibamba.grammars;

import java.util.*;

import static ru.bmstu.kibamba.grammars.GrammarUtils.*;

public class LanguageNonEmptinessChecker {

    /**
     * eliminates unnecessary non-terminals
     *
     * @param grammar input grammar to eliminate unnecessary non-terminals
     * @return Grammar with only non-terminals that can generate terminals chain
     */
    public static Grammar eliminatesUnnecessaryNonterminals(Grammar grammar) {
        Grammar clonedGrammar = grammar.clone();
        Set<String> nENonterminals = performStep01(grammar);

        clonedGrammar.getNonterminals().retainAll(nENonterminals);
        List<Production> productions = new ArrayList<>();

        for (Production production : grammar.getProductions()) {
            var chains = getProductionChainsArray(production);
            for (String chain : chains) {
                var tokens = getProductionTokenArray(chain);
                var canAddProduction = true;
                for (String token : tokens) {
                    if (!(isAlphaBelongSet(token, nENonterminals) ||
                            isAlphaBelongSet(token, grammar.getTerminals()))) {
                        canAddProduction = false;
                        break;
                    }
                }
                if (canAddProduction) {
                    productions.add(new Production(production.getNonterminal(),
chain));
                }
            }
        }

        return new Grammar(clonedGrammar.getNonterminals(),
                grammar.getTerminals(),
                productions,
                grammar.getFirstSymbol());
    }

    public static Set<String> performStep01(Grammar grammar) {
```

```
        Set<String> emptyNonterminals = new HashSet<>();
        Grammar clonedGrammar = grammar.clone();

        return performStep02(clonedGrammar, emptyNonterminals);
    }

    private static Set<String> performStep02(Grammar grammar, Set<String> predNonter-
minals) {
        Set<String> currentNonterminals = new LinkedHashSet<>();
        for (Production production : grammar.getProductions()) {
            var chains = getProductionChainsArray(production);
            for (String chain : chains) {
                if (isAlphaBelongSet(chain, predNonterminals) || isAlphaBe-
longSet(chain, grammar.getTerminals())
                        || chain.equals("£")) {
                    currentNonterminals.add(production.getNonterminal());
                    break;
                }
            }
        }
        currentNonterminals.addAll(predNonterminals);
        return performStep03(grammar, currentNonterminals, predNonterminals);
    }

    private static Set<String> performStep03(Grammar grammar, Set<String> currentNon-
terminals,
                                             Set<String> predNonterminals) {
        if (!currentNonterminals.equals(predNonterminals)) {
            return performStep02(grammar, currentNonterminals);
        } else {
            //performStep04(grammar, currentNonterminals);
            return currentNonterminals;
        }
    }

    private static void performStep04(Grammar grammar, Set<String> eNonterminals) {
        if (eNonterminals.contains(grammar.getFirstSymbol())) {
            System.out.println("ДА");
        } else {
            System.out.println("НЕТ");
        }
    }

}
```

Листинг 3 – Класс устранения недостижимых символов

```
package ru.bmstu.kibamba.grammars;

import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;

import static ru.bmstu.kibamba.grammars.GrammarUtils.*;

public class UnreachableCharacterEliminator {

    public static Grammar eliminatesUnreachableCharacter(Grammar grammarWithOnlyNec-
essaryNonterminals) {
        Grammar clonedGrammar = grammarWithOnlyNecessaryNonterminals.clone();
        Set<String> reachableCharacters = performStep01(grammarWithOnlyNecessaryNon-
terminals);
        Set<String> nonterminals = new LinkedHashSet<>(reachableCharacters);
```

```
        nonterminals.retainAll(clonedGrammar.getNonterminals());
        Set<String> terminals = new LinkedHashSet<>(reachableCharacters);
        terminals.retainAll(clonedGrammar.getTerminals());
        List<Production> productions = new ArrayList<>();

        for (Production production : clonedGrammar.getProductions()) {
            var chains = getProductionChainsArray(production);
            for (String chain : chains) {
                var canAdd = true;
                var tokens = getProductionTokenArray(chain);
                for (String token : tokens) {
                    if (!isAlphaBelongSet(token, reachableCharacters)) {
                        canAdd = false;
                        break;
                    }
                }
                if (canAdd) {
                    productions.add(new Production(production.getNonterminal(),
chain));
                }
            }
        }
        return new Grammar(nonterminals, terminals, productions, clonedGrammar.get-
FirstSymbol());
    }

    private static Set<String> performStep01(Grammar grammarWithOnlyNecessaryNonter-
minals) {
        Grammar clonedGrammar = grammarWithOnlyNecessaryNonterminals.clone();
        Set<String> initialFirstSymbols = new LinkedHashSet<>();
        initialFirstSymbols.add(clonedGrammar.getFirstSymbol());
        return performStep02(clonedGrammar, initialFirstSymbols);
    }

    private static Set<String> performStep02(Grammar clonedGrammar, Set<String>
prevFirstSymbols) {
        Set<String> currentFirstSymbols = new LinkedHashSet<>();
        for (Production production : clonedGrammar.getProductions()) {
            if (isAlphaBelongSet(production.getNonterminal(), prevFirstSymbols)) {
                var chains = getProductionChainsArray(production);
                for (String chain : chains) {
                    var tokens = getProductionTokenArray(chain);
                    currentFirstSymbols.addAll(tokens);
                }
            }
        }
        currentFirstSymbols.addAll(prevFirstSymbols);
        return performStep03(clonedGrammar, currentFirstSymbols, prevFirstSymbols);
    }

    private static Set<String> performStep03(Grammar clonedGrammar, Set<String> cur-
rentFirstSymbols,
                                             Set<String> prevFirstSymbols) {
        if (!currentFirstSymbols.equals(prevFirstSymbols)) {
            return performStep02(clonedGrammar, currentFirstSymbols);
        } else {
            return performStep04(currentFirstSymbols);
        }

    }

    private static Set<String> performStep04(Set<String> currentFirstSymbols) {
        return currentFirstSymbols;
    }
}
```

Листинг 4 – Основной класс запуски программы

```
import ru.bmstu.kibamba.files.GrammarFileReader;
import ru.bmstu.kibamba.files.GrammarFileWriter;
import ru.bmstu.kibamba.grammars.Grammar;
import ru.bmstu.kibamba.grammars.LanguageNonEmptinessChecker;
import ru.bmstu.kibamba.grammars.LeftRecursionEliminator;
import ru.bmstu.kibamba.grammars.UnreachableCharacterEliminator;

import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        String[] LEFT_RECURSION_TEST_FILENAMES = {
                "input_question1_example2_27",
                "input_question1_example4_7",
                "input_question1_example4_9",
                "input_question1_example4_11"
        };

        String[] USELESS_SYMBOLS_ELIMINATING_TEST_FILENAMES = {
                "input_question2_example_2_22",
                "input_question2_task_2_4_6"
        };
        var firstQuestionFileName = LEFT_RECURSION_TEST_FILENAMES[3];
        var leftRecursionModifiedGrammarFileName = firstQuestionFileName
                .replace("input", "output");
        var leftFactorizedGrammarFileName = firstQuestionFileName
                .replace("input", "output_left_fact");

        Grammar grammarToEliminateLeftRecursion = GrammarFileReader.readGram-
mar(firstQuestionFileName);
        GrammarFileWriter.writeGrammarJsonFile(grammarToEliminateLeftRecursion, "G0",
                firstQuestionFileName);

        Grammar leftRecursionModifiedGrammar = LeftRecursionEliminator
                .removeLeftRecursion(grammarToEliminateLeftRecursion,
                false);
        GrammarFileWriter.writeGrammar(leftRecursionModifiedGrammar, leftRecur-
sionModifiedGrammarFileName);
        GrammarFileWriter.writeGrammarJsonFile(leftRecursionModifiedGrammar, "G1",
                leftRecursionModifiedGrammarFileName);

        Grammar leftFactorizedGrammar = LeftRecursionEliminator.leftFactorsProduc-
tion(leftRecursionModifiedGrammar);
        GrammarFileWriter.writeGrammar(leftFactorizedGrammar, leftFactorizedGram-
marFileName);
        GrammarFileWriter.writeGrammarJsonFile(leftFactorizedGrammar, "G1'",
                leftFactorizedGrammarFileName);

        var secondQuestionFileName = USELESS_SYMBOLS_ELIMINATING_TEST_FILENAMES[0];
        var grammarWithOnlyUselessNonterminalsFileName = secondQuestionFileName
                .replace("input", "output_useless");
        var grammarWithOnlyReachableCharacterFileName = secondQuestionFileName
                .replace("input", "output_reachable");

        Grammar grammarToEliminateUselessCharacters = GrammarFileReader.readGram-
mar(secondQuestionFileName);
        GrammarFileWriter.writeGrammarJsonFile(grammarToEliminateUselessCharacters,
"G0",
                secondQuestionFileName);

        Grammar grammarWithOnlyUselessNonterminals = LanguageNonEmptinessChecker.
                eliminatesUnnecessaryNonterminals(grammarToEliminateUselessCharac-
ters);
```

```
        GrammarFileWriter.writeGrammar(grammarWithOnlyUselessNonterminals, grammar-
WithOnlyUselessNonterminalsFileName);
        GrammarFileWriter.writeGrammarJsonFile(grammarWithOnlyUselessNonterminals,
"G1",
                grammarWithOnlyUselessNonterminalsFileName);

        Grammar grammarWithOnlyReachableCharacters = UnreachableCharacterEliminator
                .eliminatesUnreachableCharacter(grammarWithOnlyUselessNonterminals);
        GrammarFileWriter.writeGrammar(grammarWithOnlyReachableCharacters, grammar-
WithOnlyReachableCharacterFileName);
        GrammarFileWriter.writeGrammarJsonFile(grammarWithOnlyReachableCharacters,
"G1'",
                grammarWithOnlyReachableCharacterFileName);


    }
}
```

Набор тестов и ожидаемые результаты для проверки правильности программы

1. Устранение левой рекурсии.

Пример 2.27. Пусть G_0 — наша обычная грамматика с правилами

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid a$$

Если применить к ней конструкцию леммы 2.15, то получится эквивалентная ей грамматика G' с правилами

$$E \to T \mid TE'$$
$$E' \to +T \mid +TE'$$
$$T \to F \mid FT'$$
$$T' \to *F \mid *FT'$$
$$F \to (E) \mid a \quad \square$$

Рис. 1 - Пример 2.27. [1]

Входной текстовой файл:

```
3
E T F
5
+ * ( ) a
6
E->E+T
E->T
T->T*F
T->F
F->(E)
F->a
E
```

Входной файл json:


```
{
  "name": "G0",
  "terminalSymbols": [
    {
      "name": "ADD",
      "spell": "+"
    },
    {
      "name": "MUL",
      "spell": "*"
    },
    {
      "name": "LPAREN",
      "spell": "("
    },
    {
      "name": "RPAREN",
      "spell": ")"
    },
    {
      "name": "IDENT",
      "spell": "a"
    }
  ],
  "nonterminalSymbols": [
    {
      "name": "E"
    },
    {
      "name": "T"
    },
    {
      "name": "F"
    }
  ],
  "productions": [
    {
      "lhs": {
        "name": "E"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "E"
          },
          {
            "type": "NONTERM",
            "name": "+"
          },
          {
            "type": "NONTERM",
            "name": "T"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "E"
      },
```

```
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "T"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "T"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "T"
        },
        {
          "type": "NONTERM",
          "name": "*"
        },
        {
          "type": "NONTERM",
          "name": "F"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "T"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "F"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "F"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "("
        },
        {
          "type": "NONTERM",
          "name": "E"
        },
        {
          "type": "NONTERM",
          "name": ")"
        }
      ]
    }
  },
  {
```

```
      "lhs": {
        "name": "F"
      },
      "rhs": {
        "symbol": [
          {
            "type": "TERM",
            "name": "IDENT"
          }
        ]
      }
    }
  ],
  "startSymbol": {
    "name": "E"
  }
}
```


Выходной текстовый файл – устранение левой рекурсии

```
5
E T F E' T'
5
( ) a + *
5
E -> T
E -> T E'
T -> F
T -> F T'
F -> ( E )
F -> a
E' -> + T
E' -> + T E'
T' -> * F
T' -> * F T'
E
```

Выходной текстовый файл – применение левой факторизации

```
7
E T F E' T' B C
6
( ) a + * £
7
E -> T B
T -> F C
F -> ( E )
F -> a
E' -> + T B
T' -> * F C
B -> E'
B -> £
C -> £
C -> T'
E
```

Выходной json файл – устранение левой рекурсии


```
{
  "name": "G1",
  "terminalSymbols": [
    {
      "name": "LPAREN",
      "spell": "("
```

```
    },
    {
      "name": "RPAREN",
      "spell": ")"
    },
    {
      "name": "IDENT",
      "spell": "a"
    },
    {
      "name": "ADD",
      "spell": "+"
    },
    {
      "name": "MUL",
      "spell": "*"
    }
  ],
  "nonterminalSymbols": [
    {
      "name": "E"
    },
    {
      "name": "T"
    },
    {
      "name": "F"
    },
    {
      "name": "E'"
    },
    {
      "name": "T'"
    }
  ],
  "productions": [
    {
      "lhs": {
        "name": "E"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "T"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "E"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "T"
          },
          {
            "type": "NONTERM",
            "name": "E'"
          }
        ]
      }
```

```
},
{
  "lhs": {
    "name": "T"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "F"
      }
    ]
  }
},
{
  "lhs": {
    "name": "T"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "F"
      },
      {
        "type": "NONTERM",
        "name": "T'"
      }
    ]
  }
},
{
  "lhs": {
    "name": "F"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "("
      },
      {
        "type": "NONTERM",
        "name": "E"
      },
      {
        "type": "NONTERM",
        "name": ")"
      }
    ]
  }
},
{
  "lhs": {
    "name": "F"
  },
  "rhs": {
    "symbol": [
      {
        "type": "TERM",
        "name": "IDENT"
      }
    ]
  }
},
```

```
{
  "lhs": {
    "name": "E'"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "+"
      },
      {
        "type": "NONTERM",
        "name": "T"
      }
    ]
  }
},
{
  "lhs": {
    "name": "E'"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "+"
      },
      {
        "type": "NONTERM",
        "name": "T"
      },
      {
        "type": "NONTERM",
        "name": "E'"
      }
    ]
  }
},
{
  "lhs": {
    "name": "T'"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "*"
      },
      {
        "type": "NONTERM",
        "name": "F"
      }
    ]
  }
},
{
  "lhs": {
    "name": "T'"
  },
  "rhs": {
    "symbol": [
      {
        "type": "NONTERM",
        "name": "*"
      },
```

```
          {
            "type": "NONTERM",
            "name": "F"
          },
          {
            "type": "NONTERM",
            "name": "T'"
          }
        ]
      }
    }
  ],
  "startSymbol": {
    "name": "E"
  }
}
```

Выходной json файл – применение левой факторизации

```
{
  "name": "G1'",
  "terminalSymbols": [
    {
      "name": "LPAREN",
      "spell": "("
    },
    {
      "name": "RPAREN",
      "spell": ")"
    },
    {
      "name": "IDENT",
      "spell": "a"
    },
    {
      "name": "ADD",
      "spell": "+"
    },
    {
      "name": "MUL",
      "spell": "*"
    },
    {
      "name": "IDENT",
      "spell": "£"
    }
  ],
  "nonterminalSymbols": [
    {
      "name": "E"
    },
    {
      "name": "T"
    },
    {
      "name": "F"
    },
    {
      "name": "E'"
    },
    {
      "name": "T'"
    },
    {
      "name": "B"
```

```
    },
    {
      "name": "C"
    }
  ],
  "productions": [
    {
      "lhs": {
        "name": "E"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "T"
          },
          {
            "type": "NONTERM",
            "name": "B"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "T"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "F"
          },
          {
            "type": "NONTERM",
            "name": "C"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "F"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "("
          },
          {
            "type": "NONTERM",
            "name": "E"
          },
          {
            "type": "NONTERM",
            "name": ")"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "F"
      },
```

```
    "rhs": {
      "symbol": [
        {
          "type": "TERM",
          "name": "IDENT"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "E'"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "+"
        },
        {
          "type": "NONTERM",
          "name": "T"
        },
        {
          "type": "NONTERM",
          "name": "B"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "T'"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "*"
        },
        {
          "type": "NONTERM",
          "name": "F"
        },
        {
          "type": "NONTERM",
          "name": "C"
        }
      ]
    }
  },
  {
    "lhs": {
      "name": "B"
    },
    "rhs": {
      "symbol": [
        {
          "type": "NONTERM",
          "name": "E'"
        }
      ]
    }
  },
  {
```

```
      "lhs": {
        "name": "B"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "£"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "C"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "£"
          }
        ]
      }
    },
    {
      "lhs": {
        "name": "C"
      },
      "rhs": {
        "symbol": [
          {
            "type": "NONTERM",
            "name": "T'"
          }
        ]
      }
    }
  ],
  "startSymbol": {
    "name": "E"
  }
}
```


$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow T * F \mid F$$
$$F \longrightarrow (E) \mid a$$

Пример 4.7. Повторенная здесь нелеворекурсивная грамматика для выражений (4.2)

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +T\ E' \mid \epsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \epsilon \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$

Рис. 2 - Пример 4.7. [2]

Входной текстовой файл:

```
3
E T F
5
+ * ( ) a
6
E->E+T
E->T
T->T*F
T->F
F->(E)
F->a
E
```

Выходной текстовый файл – устранение левой рекурсии

```
5
E E' T T' F
6
+ £ * ( ) id
5
E -> T E'
E' -> + T E'
E' -> £
T -> F T'
T' -> * F T'
T' -> £
F -> ( E )
F -> id
E
```

Выходной текстовый файл – применение левой факторизации

```
5
E E' T T' F
6
+ £ * ( ) id
5
E -> T E'
E' -> + T E'
E' -> £
T -> F T'
T' -> * F T'
T' -> £
F -> ( E )
F -> id
E
```

двумя или более шагами порождения. Рассмотрим, например, грамматику

$$\begin{aligned} S &\rightarrow A\,a \mid b \\ A &\rightarrow A\,c \mid S\,d \mid \epsilon \end{aligned} \tag{4.11}$$

Пример 4.9. Применим алгоритм 4.8 к грамматике (4.11). Технически из-за наличия ϵ-продукции алгоритм может не работать, но в данном случае продукция $A \rightarrow \epsilon$ не мешает работе.

Мы располагаем нетерминалы в порядке S, A. Непосредственной левой рекурсии среди S-продукций нет, так что в процессе работы внешнего цикла при $i = 1$ ничего не происходит. При $i = 2$ мы подставляем S-продукцию в $A \rightarrow S\,d$ для получения следующих A-продукций:

$$A \rightarrow A\,c \mid A\,a\,d \mid b\,d \mid \epsilon$$

Устранение непосредственной левой рекурсии среди A-продукций дает грамматику

$$\begin{aligned} S &\rightarrow A\,a \mid b \\ A &\rightarrow b\,d\,A' \mid A' \\ A' &\rightarrow c\,A' \mid a\,d\,A' \mid \epsilon \end{aligned}$$

□

Рис. 3 - Пример 4.9. [2]

Входной текстовой файл:

```
2
S A
5
a c b d £
5
S -> A a
S -> b
A -> A c
A -> S d
A -> £
S
```

Выходной текстовый файл – устранение левой рекурсии

```
3
S A A'
5
```

```
a b d c £
3
S -> A a
S -> b
A -> b d A'
A -> A'
A' -> c A'
A' -> a d A'
A' -> £
S
```

Выходной текстовый файл – применение левой факторизации

```
3
S A A'
5
a b d c £
3
S -> A a
S -> b
A -> b d A'
A -> A'
A' -> c A'
A' -> a d A'
A' -> £
S
```

Пример 4.11. Следующая грамматика абстрагирует проблему "висящего **else**":

$$\begin{aligned} S &\rightarrow i\,E\,t\,S \mid i\,E\,t\,S\,e\,S \mid a \\ E &\rightarrow b \end{aligned} \tag{4.12}$$

Здесь i, t и e означают **if**, **then** и **else**, E и S соответствуют "условному выражению" и "инструкции". Будучи левофакторизованной, эта грамматика принимает следующий вид:

$$\begin{aligned} S &\rightarrow i\,E\,t\,S\,S' \mid a \\ S' &\rightarrow e\,S \mid \epsilon \\ E &\rightarrow b \end{aligned} \tag{4.13}$$

Таким образом, можно расширить S до $iEtSS'$ для входного i и подождать, пока из входного потока не будет считано $iEtS$, чтобы затем решить, расширять ли S' до eS или до ϵ. Конечно, обе эти грамматики неоднозначны и при обнаружении во входном потоке e будет неясно, какая из альтернатив для S' должна быть выбрана. В примере 4.19 обсуждается путь решения этой дилеммы. □

Рис. 4 - Пример 4.11. [2]

Входной текстовой файл:

```
2
S E
5
i t e a b
4
S->iEtS
S->iEtSeS
```

```
S->a
E->b
S
```

Выходной текстовый файл – устранение левой рекурсии

```
2
S E
5
i t e a b
2
S -> i E t S
S -> i E t S e S
S -> a
E -> b
S
```

Выходной текстовый файл – применение левой факторизации

```
3
S E S'
6
i t a b £ e
3
S -> i E t S S'
S -> a
E -> b
S' -> £
S' -> e S
S
```

2. Устранение бесполезных символов

Пример 2.22. Рассмотрим грамматику $G=(\{S, A, B\}, \{a, b\}, P, S)$, где P состоит из правил

$$S \longrightarrow a \mid A$$
$$A \longrightarrow AB$$
$$B \longrightarrow b$$

Применим к G алгоритм 2.9. На шаге (1) получим $N_e=\{S, B\}$ и $G_1=(\{S, B\}, \{a, b\}, \{S \longrightarrow a, B \longrightarrow b\}, S)$. Применив алгоритм 2.8, получим $V_2=V_1=\{S, a\}$. Итак, $G'=(\{S\}, \{a\}, \{S \longrightarrow a\}, S)$.

Если применить к G сначала алгоритм 2.8, то окажется, что все символы достижимы, так что грамматика не изменится. Затем применение алгоритма 2.7 дает $N_e=\{S, B\}$, и результирующей будет грамматика G_1, отличная от G'. □

Часто бывает удобно устранить из КС-грамматики G e-правила, т. е. правила вида $A \longrightarrow e$. Но если $e \in L(G)$, то очевидно, что без правил вида $A \longrightarrow e$ не обойтись.

Рис. 5 - Пример 2.22 [1]

Входной текстовой файл:

```
3
S A B
2
a b
4
```

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

Выходная грамматика G1

```
2
S B
2
a b
2
S -> a
B -> b
S
```

Выходная грамматика G’ – содержит только полезные символы входной грамматики

```
1
S
1
a
1
S -> a
S
```

Выходная грамматика G’ – содержит только полезные символы входной грамматики (json file)

```
{
  "name": "G'",
  "terminalSymbols": [
    {
      "name": "IDENT",
      "spell": "a"
    }
  ],
  "nonterminalSymbols": [
    {
      "name": "S"
    }
  ],
  "productions": [
    {
      "lhs": {
        "name": "S"
      },
      "rhs": {
        "symbol": [
          {
            "type": "TERM",
            "name": "IDENT"
          }
        ]
      }
    }
  ],
  "startSymbol": {
    "name": "S"
  }
}
```

2.4.6. Преобразуйте грамматику

$$S \to A \mid B$$
$$A \to aB \mid bS \mid b$$
$$B \to AB \mid Ba$$
$$B \to AS \mid b$$

в эквивалентную КС-грамматику, не содержащую бесполезных символов.

Рис. 6 - Упражнение 2.4.6. [1]

Входной текстовой файл:

```
3
S A B
2
a b
9
S->A
S->B
A->aB
A->bS
A->b
B->AB
B->Ba
B->AS
B->b
S
```

Выходная грамматика G1

```
3
S A B
2
a b
9
S -> A
S -> B
A -> a B
A -> b S
A -> b
B -> A B
B -> B a
B -> A S
B -> b
S
```

Выходная грамматика G’ – содержит только полезные символы входной грамматики

```
3
A B S
2
a b
9
S -> A
S -> B
A -> a B
A -> b S
A -> b
B -> A B
B -> B a
```

```
B -> A S
B -> b
S
```