МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Институт информатики, математики и робототехники

Отчёт по лабораторным работам по дисциплине «Теория автоматов и формальных языков»

Вариант №13

Выполнил:
Студент гр. ПРО-341Б
Репин Д.А.

Проверила:
Абдрахманова Р.П.

Уфа — 2024

Содержание

1 Введение **3**

2 Ход работы **4**

2.1 Задание 1 4

2.2 Задание 2 5

2.3 Задание 3 5

2.4 Задание 4 6

2.5 Задание 5 7

2.6 Задание 6 8

2.7 Задание 7 9

2.8 Задание 8 11

2.9 Задание 9 12

2.10 Задание 10 13

2.11 Задание 11 15

2.12 Задание 12 17

2.13 Приложение 19

3 Выводы **20**

4 Список литературы **20**

1 Введение

Цель работы

Автоматизировать расчетную работу при работе с формальными грамматиками и конечными автоматами

Задачи работы

- Решить предложенные задания в соответствии с вариантом работы №13
- Разработать программы для автоматизиации процесса
- Оформить отчет по проделанной работе

2 Ход работы

2.1 Задание 1

Задача: Для данных языков L_1 и L_2 найти...

Универсум в условии задан не был, но следует полагать, что $V = \{a, b\}$, а значит $V^* = \{\Lambda, a, b, aa, ab, ba \ldots\}$

```cpp
#include <iostream>
#include "grammar/grammar.hpp"
#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;
  PrintHead("--- 1 ---");

  Alphabet alphabet = {'a', 'b'};
  Language language_1 = {"a", "aa", "baa"};
  Language language_2 = {"b", "aa", "abb"};

  PrintTask("L1 U L2" ,GetChains(Union(language_1, language_2)));
  PrintTask("L1  L2", GetChains(Intersection(language_1, language_2)));

  PrintTask("L1 \\ L2", GetChains(Difference(language_1, language_2)));
  PrintTask("L2 \\ L1", GetChains(Difference(language_2, language_1)));

  PrintTask("L1L2", GetChains(Concatenation(language_1, language_2)));
  PrintTask("L2L1", GetChains(Concatenation(language_2, language_1)));

  PrintTask("L1'", GetChains(Complement(language_1, alphabet)));
  PrintTask("L2'", GetChains(Complement(language_2, alphabet)));

  return EXIT_SUCCESS;
}
```

Листинг 1: Код к задаче №1

```
--- №1 ---
L1 ∪ L2 = {a, b, aa, abb, baa}
L1 ∩ L2 = {aa}
L1 \ L2 = {a, baa}
L2 \ L1 = {b, abb}
L1L2 = {ab, aaa, aab, aaaa, aabb, baab, aaabb, baaaa, baaabb}
L2L1 = {ba, aaa, baa, aaaa, abba, bbaa, aabaa, abbaa, abbbaa}
L1' = {人, b, ab, ba, bb, aaa, aab, aba, abb, bab, bba, bbb,
 baba, babb, bbaa, bbab, bbba, bbbb, ...}
L2' = {人, a, ab, ba, bb, aaa, aab, aba, baa, bab, bba, bbb,
 baba, babb, bbaa, bbab, bbba, bbbb, ...}
```

Рис. 1: Вывод программы к задаче №1

2.2 Задание 2

Задача: Для данного языка L найти усеченное замыкание Клини, замыкание Клини

```cpp

#include <iostream>

#include "grammar/grammar.hpp"
#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;

  PrintHead("--- 2 ---");

  Language language_3 = {"01", "010101"};

  PrintTask("L*",GetChains(KleeneStar(language_3, 2)));
  PrintTask("L+",GetChains(KleenePlus(language_3, 2)));

  return EXIT_SUCCESS;
}
```

Листинг 2: Код к задаче №2

```
--- №2 ---
L* = {人, 01, 0101, 010101, 01010101, 010101010101, ...}
L+ = {01, 0101, 010101, 01010101, 010101010101, ...}
```

Рис. 2: Вывод программы к задаче №2

2.3 Задание 3

Задача: Для данной грамматики $G = \langle V_N, V_t, P, S \rangle$ определить:

1. Класс грамматики
2. язык $L(G)$, порождаемый данной грамматикой.

Грамматика контекстно-свободная

```cpp
#include <iostream>

#include "grammar/grammar.hpp"

#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;
```

```
9
10    PrintHead("--- 3 ---");
11
12    Grammar grammar({
13                        {"S", {"abcA"}},
14                        {"A", {"bbbS", ""}}
15                      },
16                      'S');
17
18
19    PrintTask("L(G)", GetChains(grammar.GetChains(10)));
20
21    return EXIT_SUCCESS;
22  }
```

Листинг 3: Код к задаче №3

```
--- №3 ---
L(G) = {abc, abcbbbabc, abcbbbabcbbbabc, abcbbbabcbbbabcbbbabc, abcbbbabcbbbabcbbbabcbbbabc, abcbbbabcbbbabcbbbabcbbbabcbbbabc,
bbbabc, abcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabc, abcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabc, abcbbbabcbbbabcbbbabcbbb
abcbbbabcbbbabcbbbabcbbbabc, abcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabcbbbabc, ... }
```

Рис. 3: Вывод программы к задаче №3

2.4 Задание 4

Задача: Построить грамматику $G = \langle V_N, V_t, P, S \rangle$ заданного класса, порождающую заданный язык $L(G)$ при данных V_N, V_t

```
1  #include <iostream>
2
3  #include "grammar/grammar.hpp"
4
5  #include "utils/format/formatter.hpp"
6
7  int main() {
8    using namespace lab::format;
9
10   PrintHead("---   4   ---");
11
12   Grammar grammar_4(
13     {
14       {"S", {"", "A", "B"}},
15       {"A", {"0A", "0"}},
16       {"B", {"1B", "1"}}
17     },
18     'S');
19
20   PrintTask("L(G)" , GetChains(grammar_4.GetChains(20)));
21
22   return EXIT_SUCCESS;
23 }
```

Листинг 4: Код к задаче №4

```
--- №4 ---
L(G) = {ε, 0, 1, 00, 11, 000, 111, 0000, 1111, 00000, 11111, 000000, 111111, 0000000, 1111111, 00000000, 11111111, 0000
00000, 111111111, 1111111111, ...}
```

Рис. 4: Вывод программы к задаче №4

2.5 Задание 5

Задача: Для заданных $|K|, |F|, V_T$, построить формальное и графическое представление детерминированного конечного автомата $M = \langle K, V_T, t, k_1, F \rangle$, принимающего заданное множество цепочек $L(M) \subset V_T^*$.

При заданных $|K|$ и $|F|$ автомат к языку $L(M)$ построить невозможно, характеристики изменены на $|K| = 4$, $|F| = 1$, $L(M) = \{ab^m bc^n\}_{m=0,n=1}$

```cpp
#include <iostream>

#include "automata/dfa/dfa.hpp"
#include "grammar/grammar.hpp"

#include "utils/format/formatter.hpp"

int main() {
    using namespace lab::format;
  using namespace std::literals;

  PrintHead("--- 5 ---");

  std::unordered_set states = {"S"s, "A"s, "B"s, "C"s};
  std::unordered_set final_states = {"C"s};
  std::unordered_map<std::string, std::unordered_map<char, std::string> >
   transitions = {
    {"S", {{'a', "A"}}},
    {"A", {{'b', "B"}}},
    {"B", {{'b', "B"}, {'c', "C"}}},
    {"C", {{'c', "C"}}}
  };

  DFA dfa("S", states, final_states, transitions);

  auto chains = dfa.GenerateChains(5);
  PrintTask("L(M)", GetChains(chains));

  dfa.ToDot("dfa.dot");
  PrintTask("Saved to : ", "dfa.dot");

  return EXIT_SUCCESS;
}
```

Листинг 5: Код к задаче №5

```
--- №5 ---
L(M) = {abc, abbc, abcc, abbbc, abbcc, abccc}
Графическое представление:  = dfa.dot
```

Рис. 5: Вывод программы к задаче №5

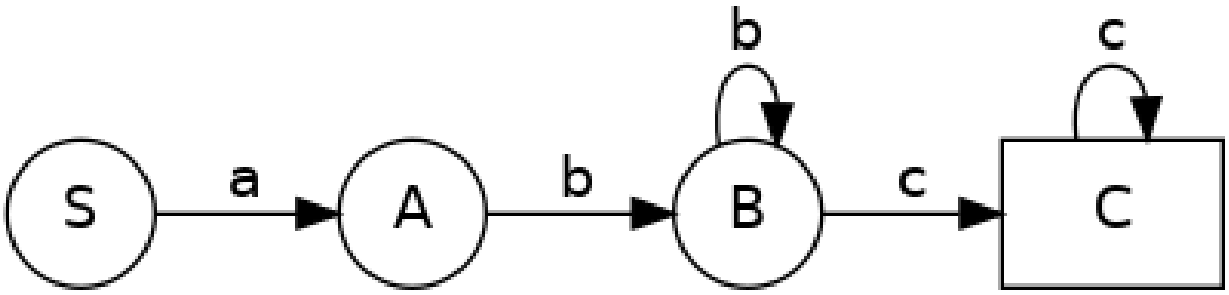


Рис. 6: Граф к задаче №5

2.6 Задание 6

Задача: Для заданных $|K|, |F|, V_T$, построить формальное и графическое представление недетерминированного конечного автомата $M = \langle K, V_T, t, k_1, F\ rangle$, принимающего заданное множество цепочек $L(M) \subset V_T^*$.

```cpp
using namespace lab::format;
 using namespace std::literals;

 PrintHead("--- 6 ---");

 std::unordered_set states = {"k1"s, "k2"s, "k3"s};
 std::unordered_set final_states = {"k3"s};
 std::unordered_map<std::string, std::unordered_map<char, std::vector<std
  ::string>>> transitions = {
   {"k1", {{'x', {"k1", "k2", "k3"}}, {'0', {}}}},
   {"k2", {{'x', {}}, {'0', {"k2", "k3"}}}},
   {"k3", {{'x', {"k3"}}, {'0', {}}}}
 };

 NFA nfa("k1", states, final_states, transitions);

 auto chains = nfa.GenerateChains(5);
 PrintTask("L(M)", GetChains(chains));

 nfa.ToDot("nfa.dot");
 PrintTask("Saved to: ", "nfa.dot");

 return EXIT_SUCCESS;
}
```

Листинг 6: Код к задаче №6

```
--- №6 ---
L(M) = {x, x0, xx, x00, x0x, xx0, xxx, x000, x00x, x0xx, xx00, xx0x, xxx0, xxxx, x0000, x000x, x00xx, x0xxx, xx000, xx00
x, xx0xx, xxx00, xxx0x, xxxx0, xxxxx}
Графическое представление:  = nfa.dot
```

Рис. 7: Вывод программы к задаче №6

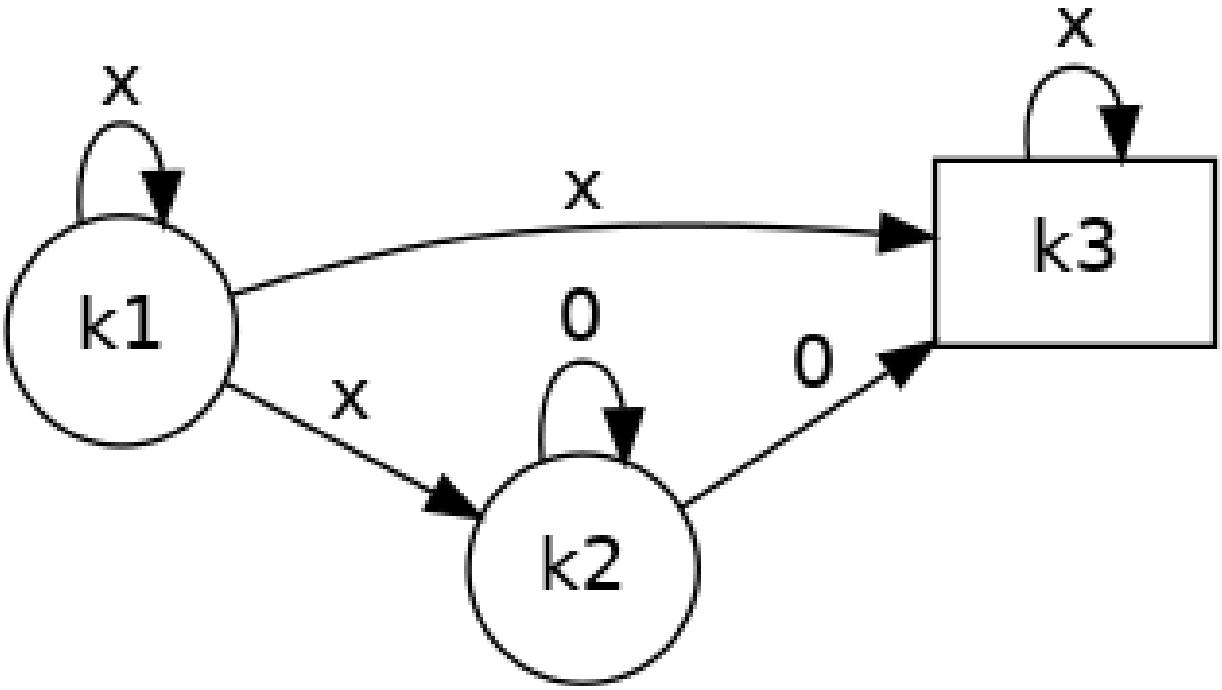


Рис. 8: Граф к задаче №6

2.7 Задание 7

Задача: Для детерминированного конечного автомата задания 5 построить формальное и графическое представление эквивалентного недетерминированного конечного автомата.

```cpp
#include <assert.h>
#include <iostream>

#include "automata/converter.h"
#include "automata/dfa/dfa.hpp"
#include "automata/nfa/nfa.hpp"
#include "grammar/grammar.hpp"

#include "utils/format/formatter.hpp"

int main() {
    using namespace lab::format;
  using namespace std::literals;

  PrintHead("---   7   ---");

  std::unordered_set states = {"S"s, "A"s, "B"s, "C"s};
```

```
18    std::unordered_set final_states = {"C"s};
19    std::unordered_map<std::string, std::unordered_map<char, std::string> >
       transitions = {
20      {"S", {{'a', "A"}}},
21      {"A", {{'b', "B"}}},
22      {"B", {{'b', "B"}, {'c', "C"}}},
23      {"C", {{'c', "C"}}}
24    };
25
26    DFA dfa("S", states, final_states, transitions);
27
28    NFA nfa = ConvertDfaToNfa(dfa);
29
30
31    assert(dfa.GenerateChains(10) == nfa.GenerateChains(10));
32    PrintTask("L(M) = L(M')", GetChains(nfa.GenerateChains(5)));
33
34    nfa.ToDot("nfa.dot");
35    PrintTask("Saved to : ", "nfa.dot");
36
37    return EXIT_SUCCESS;
38  }
```

Листинг 7: Код к задаче №7

```
--- №7 ---
L(M) = L(M') = {abc, abbc, abcc, abbbc, abbcc, abccc}
Графическое представление:  = nfa.dot
```

Рис. 9: Вывод программы к задаче №7

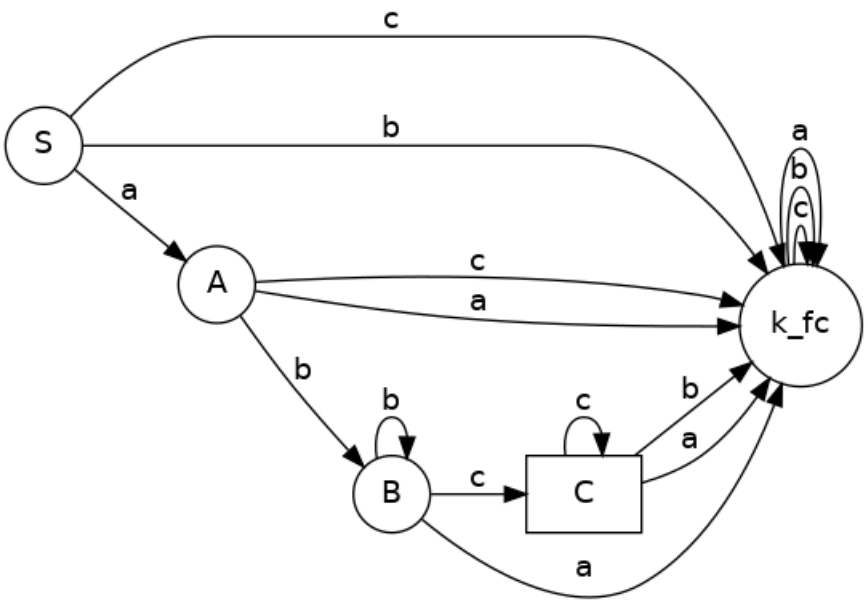


Рис. 10: Граф к задаче №7

2.8 Задание 8

Задача: Для недетерминированного конечного автомата задания 6 построить формальное и графическое представление эквивалентного детерминированного конечного автомата.

```cpp
#include <cassert>
#include <iostream>
#include "automata/dfa/dfa.hpp"
#include "automata/nfa/nfa.hpp"
#include "grammar/grammar.hpp"

#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;
  using namespace std::literals;

  PrintHead("---   8   ---");

  std::unordered_set states = {"k1"s, "k2"s, "k3"s};
  std::unordered_set final_states = {"k3"s};
  std::unordered_map<std::string, std::unordered_map<char, std::vector<std
    ::string>>> transitions = {
    {"k1", {{'x', {"k1", "k2", "k3"}}, {'0', {}}}},
    {"k2", {{'x', {}}, {'0', {"k2", "k3"}}}},
    {"k3", {{'x', {"k3"}}, {'0', {}}}}
  };

  NFA nfa("k1", states, final_states, transitions);

  std::unordered_set states_1 = {"{k1}"s, "{k1, k2, k3}"s, "{k2, k3}"s, "{
    k3}"s, empty_set};
  std::unordered_set final_states_1 = { "{k1, k2, k3}"s, "{k2, k3}"s,"{k3}
    "s};

  std::unordered_map<std::string, std::unordered_map<char, std::string>>
    transitions_1= {
      {"{k1}", {{'x', "{k1, k2, k3}"}, {'0', empty_set}}},
      {"{k1, k2, k3}", {{'x', "{k1, k2, k3}"}, {'0', "{k2, k3}"}}},
      {"{k2, k3}", {{'x', "{k3}"}, {'0', "{k2, k3}"}}},
      {"{k3}", {{'x', "{k3}"}, {'0', empty_set}}},
      {empty_set, {{'x', empty_set}, {'0', empty_set}}}
  };

  DFA dfa("{k1}", states_1, final_states_1, transitions_1);

  assert(dfa.GenerateChains(10) == nfa.GenerateChains(10));
  PrintTask("L(M')", GetChains(dfa.GenerateChains(5)));
  PrintTask("L(M)", GetChains(nfa.GenerateChains(5)));

  dfa.ToDot("dfa.dot");
  PrintTask("Saved to ", "dfa.dot");

  return EXIT_SUCCESS;
}
```

Листинг 8: Код к задаче №8

```
--- №8 ---
L(M') = {x, x0, xx, x00, x0x, xx0, xxx, x000, x00x, x0xx, xx00, xx0x, xxx0, xxxx, x0000, x000x, x00xx, x0xxx, xx000, xx0
0x, xx0xx, xxx00, xxx0x, xxxx0, xxxxx}
L(M) = {x, x0, xx, x00, x0x, xx0, xxx, x000, x00x, x0xx, xx00, xx0x, xxx0, xxxx, x0000, x000x, x00xx, x0xxx, xx000, xx00
x, xx0xx, xxx00, xxx0x, xxxx0, xxxxx}
Графическое представление:  = dfa.dot
```

Рис. 11: Вывод программы к задаче №8

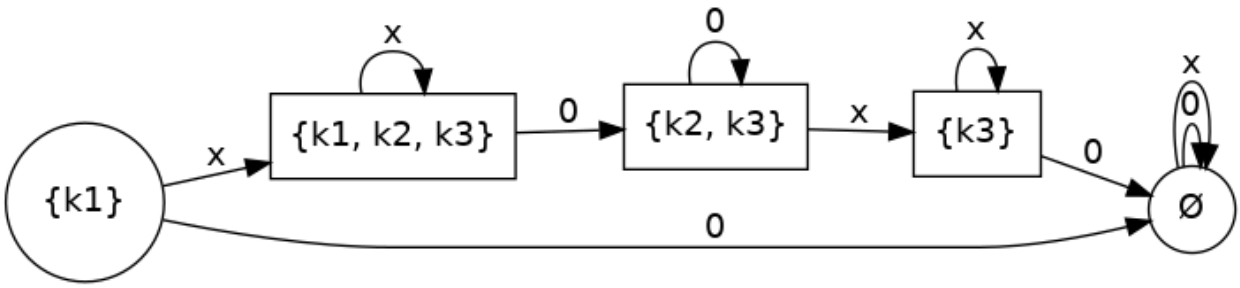


Рис. 12: Граф к задаче №8

2.9 Задание 9

Задача: Для детерминированного конечного автомата задания 5 построить регулярную грамматику, порождающую язык, совпадающий с множеством цепочек, принимаемых данным автоматом.

```cpp
#include <iostream>

#include "automata/dfa/dfa.hpp"
#include "grammar/grammar.hpp"

#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;
  using namespace std::literals;

  PrintHead("---   9   ---");

  std::unordered_set states = {"S"s, "A"s, "B"s, "C"s};
  std::unordered_set final_states = {"C"s};
  std::unordered_map<std::string, std::unordered_map<char, std::string> >
   transitions = {
    {"S", {{'a', "A"}}},
    {"A", {{'b', "B"}}},
    {"B", {{'b', "B"}, {'c', "C"}}},
    {"C", {{'c', "C"}}}
  };

  DFA dfa("S", states, final_states, transitions);
  auto grammar_auto = dfa.GetRegularGrammar();

  PrintTask("L(G)", GetChains(grammar_auto.GetChains(20)));

  auto [Vn, Vt, S, P] = grammar_auto.GetFormalRepresentation();
```

```
29
30    PrintTask("Vn", GetChains(Vn));
31
32
33    std::set<std::string> Vt_str;
34
35    for (const auto& terminal : Vt) {
36      Vt_str.insert(std::string(1, terminal));
37    }
38
39    PrintTask("Vt", GetChains(Vt_str));
40    PrintTask("S", std::string(1,S));
41
42    std::stringstream ss_2;
43
44    for (const auto& [lhs, rhs_set] : P) {
45      for (const auto& rhs : rhs_set) {
46        ss_2 << lhs << " -> " << rhs << "; ";
47      }
48    }
49
50    PrintTask("P", "{" + ss_2.str() + "}");
51
52    return EXIT_SUCCESS;
53  }
```

Листинг 9: Код к задаче №9

```
--- №9 ---
L(G) = {abc, abbc, abcc, abbbc, abbcc, abccc, abbbbc, abbbcc, abbccc,
c, abbbbbbc, abbbbbcc, abbbbccc, abbbcccc, abbccccc, ...}
Vn = {A, B, C, S}
Vt = {a, b, c}
S = S
P = {B → bB; B → c; B → cC; C → c; C → cC; A → bB; S → aA; }
```

Рис. 13: Вывод программы к задаче №9

2.10 Задание 10

Задача: Для данной регулярной грамматики построить недетерминированный конечный автомат, принимающий множество цепочек, совпадающее с языком данной грамматики.

```
1  #include <iostream>
2
3  #include "automata/converter.hpp"
4  #include "grammar/grammar.hpp"
5
6  #include "utils/format/formatter.hpp"
7
8  int main() {
9    using namespace lab::format;
10
```

```
  PrintHead("--- 10 ---");

  Grammar grammar_4(
    {
      {"S", {"a", "b", "c","aS","cS"}}
    },
    'S');

  PrintTask("L(G)", GetChains(grammar_4.GetChains(45)));
  auto nfa = GetNfaFromGrammar(grammar_4);

  PrintTask("L(G)", GetChains(nfa.GenerateChains(4)));


  nfa.ToDot("nfa.dot");
  PrintTask("Saved to ", "nfa.dot");

  return EXIT_SUCCESS;
}
```

Листинг 10: Код к задаче №10

```
--- %10 ---
L(G) = {a, b, c, aa, ab, ac, ca, cb, cc, aaa, aab, aac, aca, acb, acc, caa, cab, cac, cca, ccb, ccc, aaaa, aaab, aaac, a
aca, aacb, aacc, acaa, acab, acac, acca, accb, accc, caaa, caab, caac, caca, cacb, cacc, ccaa, ccab, ccac, ccca, cccb, c
ccc, ...}
L(G) = {a, b, c, aa, ab, ac, ca, cb, cc, aaa, aab, aac, aca, acb, acc, caa, cab, cac, cca, ccb, ccc, aaaa, aaab, aaac, a
aca, aacb, aacc, acaa, acab, acac, acca, accb, accc, caaa, caab, caac, caca, cacb, cacc, ccaa, ccab, ccac, ccca, cccb, c
ccc}
```

Рис. 14: Вывод программы к задаче №10

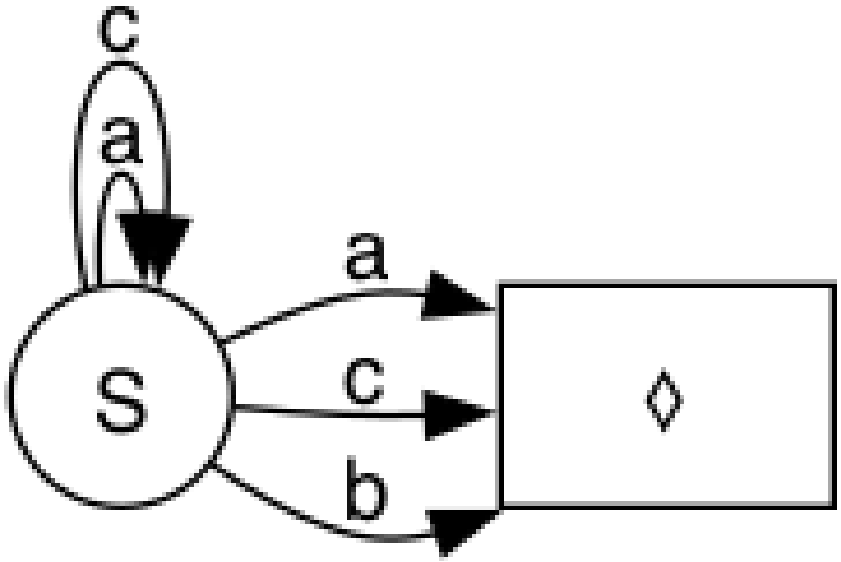


Рис. 15: Граф к задаче №10

2.11 Задание 11

Задача:

1. В грамматике задания 4 построить все возможные выводы для цепочки наименьшей длины из языка данной грамматики и все соответствующие размеченные выводы.
2. Для полученных выводов и размеченных выводов построить соответствующие деревья выводов и растянутые деревья выводов.
3. Определить, однозначна ли рассматриваемая грамматика.

```cpp
#include <iostream>
#include <optional>
#include <map>

#include "grammar/grammar.hpp"
#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;

 Productions rules = {
      {"S", {"1", "A", "B"}},
      {"A", {"0A", "0"}},
      {"B", {"1B", "1"}},
  };

  std::vector<char> non_terminals{'S', 'A', 'B'};
  std::vector<char> terminals{'0', '1', '1'};

  Grammar grammar(rules, 'S', {terminals.begin(), terminals.end()});

  auto result = grammar.GenerateMarkedDerivations(5, 'S');

  std::string target = "000";

  for (const auto& path : result) {
    if (path.at(path.size() - 1) == target) {
      for (const auto& chain : path) {
        std::cout << chain << " ";
      }
      std::cout << std::endl;
    }
  }

  return EXIT_SUCCESS;
}
```

Листинг 11: Код к задаче №10

1. Для пустой цепочки вывод тривиален: $\Delta_1 = (S, \Lambda)$ $\Delta'_1 = (*S*, \Lambda)$
   Вывод программы для цепочки 000: *S* *A* 0*A* 00*A* 000

2. Деревья вывода

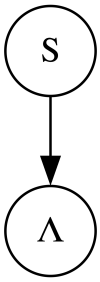
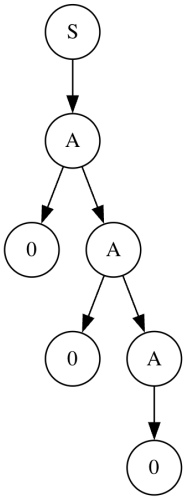
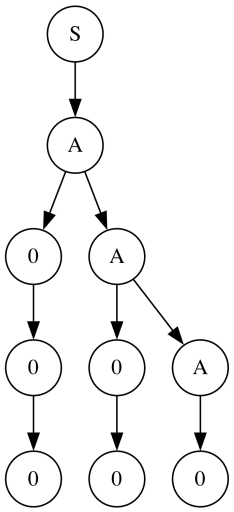


Рис. 16: Дерево вывода для минимальной цепочки



(a) Дерево вывода



(b) Растянутое дерево вывода

Рис. 17: Деревья вывода для цепочки 000

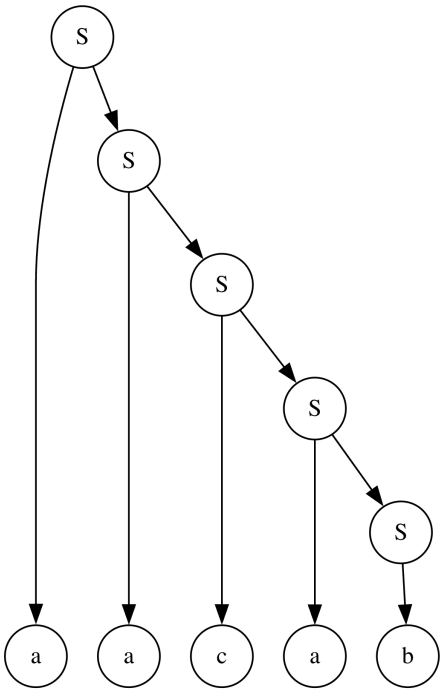
3. Глядя на продукции грамматики G, совершенно нетрудно заметить, что каждой цепочке языка соответствует единственное дерево вывода, что, в соответствии с определением, говорит об однозначности грамматики G.

2.12 Задание 12

Задача:

1. Проверить, является ли грамматика задания 10 грамматикой простого предшествования.
2. При положительном ответе применить синтаксический анализатор для грамматики простого предшествования по схеме «снизу вверх» для построения дерева вывода любой цепочки из языка данной грамматики.

1. Является грамматикой простого предшествования
2. Пусть x = aacab



$$L(G) = \{\{a, c\}^* b\} \cup \{a, c\}^+$$

```cpp
#include <iostream>
#include <optional>
#include <map>

#include "grammar/grammar.hpp"
#include "utils/format/formatter.hpp"

int main() {
  using namespace lab::format;
```

```
10
11      {
12       Productions rules = {
13            {"S", {"a", "b", "c", "aS", "cS"}}
14        };
15
16        std::vector<char> non_terminals{'S'};
17        std::vector<char> terminals{'a', 'b', 'c'};
18        std::vector<char> both{non_terminals.begin(), non_terminals.end()};
19
20        for (auto term : terminals) {
21          both.push_back(term);
22        }
23
24        Grammar grammar(rules, 'S', {terminals.begin(), terminals.end()});
25
26        auto relations = CheckGPP(rules, terminals, non_terminals, grammar);
27
28        if (!relations.has_value()) {
29          std::cout << "Not the simple precedence grammar!" << std::endl;
30        } else {
31          std::cout << "   ";
32
33          for (auto symbol : both) {
34            std::cout << symbol << " ";
35          }
36
37          std::cout << std::endl;
38
39          for (auto [symbol, symbol_relations] : (*relations)) {
40            std::cout << symbol << " ";
41
42            for (auto [second_symbol, relation] : symbol_relations) {
43              switch (relation) {
44                case Relation::EQUAL:
45                  std::cout << "=" << " ";
46                  break;
47                case Relation::BEFORE:
48                  std::cout << "<" << " ";
49                  break;
50                case Relation::AFTER:
51                  std::cout << ">" << " ";
52                  break;
53                case Relation::NONE:
54                  std::cout << "- ";
55                  break;
56              }
57            }
58            std::cout << std::endl;
59          }
60        }
61      }
62
63
64
65    return EXIT_SUCCESS;
66  }
```

Листинг 12: Код программы задания №12

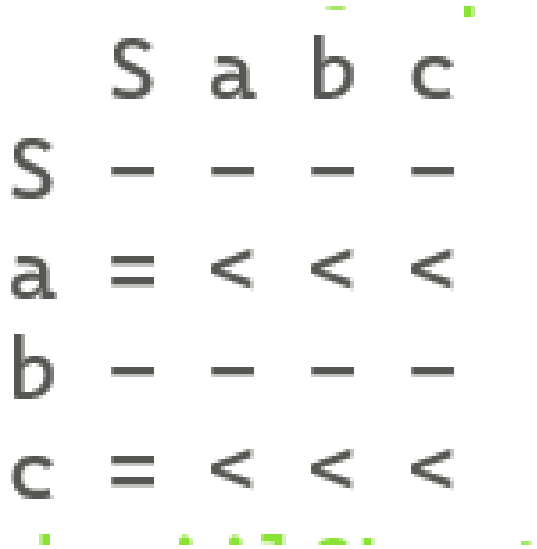


Рис. 18: Вывод программы к задаче №12

2.13 Приложение

Исходный код библиотеки для работы с формальными языками и конечными автоматами, разработанной в ходе выполнения лабораторных работ, располагается в репозитории по ссылке **github.com/Repin-Daniil/formal-language-and-automata-theory**

3 Выводы

В рамках лабораторной работы была разработана библиотека для работы с формальными языками и конечными автоматами, решены предложенные задачи в соответствии с вариантом №13

4 Список литературы

1. Задания расчетно-графической работы по дисциплине «Теория автоматов и формальных языков» — Уфа: УГАТУ, 2012. — 11с

2. Орехов Ю.В., Ефремова А.Н., Орехов Э.Ю. Основы теории формальных языков: учебное пособие — Уфа: УГАТУ, 2014 — 124с