

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»  
(БГТУ им. В.Г. Шухова)**



**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ**

**Лабораторная работа №1**

по дисциплине: Теория автоматов и формальных языков  
тема: «Формальные грамматики. Выводы»

Выполнил: ст. группы ПВ-233  
Мовчан Антон Юрьевич

Проверили:  
ст. пр. Рязанов Юрий Дмитриевич

Белгород 2025 г.

## Лабораторная работа №1

Цель работы: изучить основные понятия теории формальных языков и грамматик.

### Вариант 8

1.  $S \rightarrow aSbA$
2.  $S \rightarrow aB$
3.  $S \rightarrow A$
4.  $A \rightarrow aAbS$
5.  $A \rightarrow aB$
6.  $A \rightarrow S$
7.  $B \rightarrow b$
8.  $B \rightarrow aA$

1. Найти терминальную цепочку  $\alpha$ ,  $|\alpha| > 10$ , для которой существует не менее двух левых выводов в заданной КС-грамматике (см. варианты заданий). Записать различные левые выводы этой цепочки. Построить деревья вывода. Определить последовательности правил, применяемые при этих выводах.

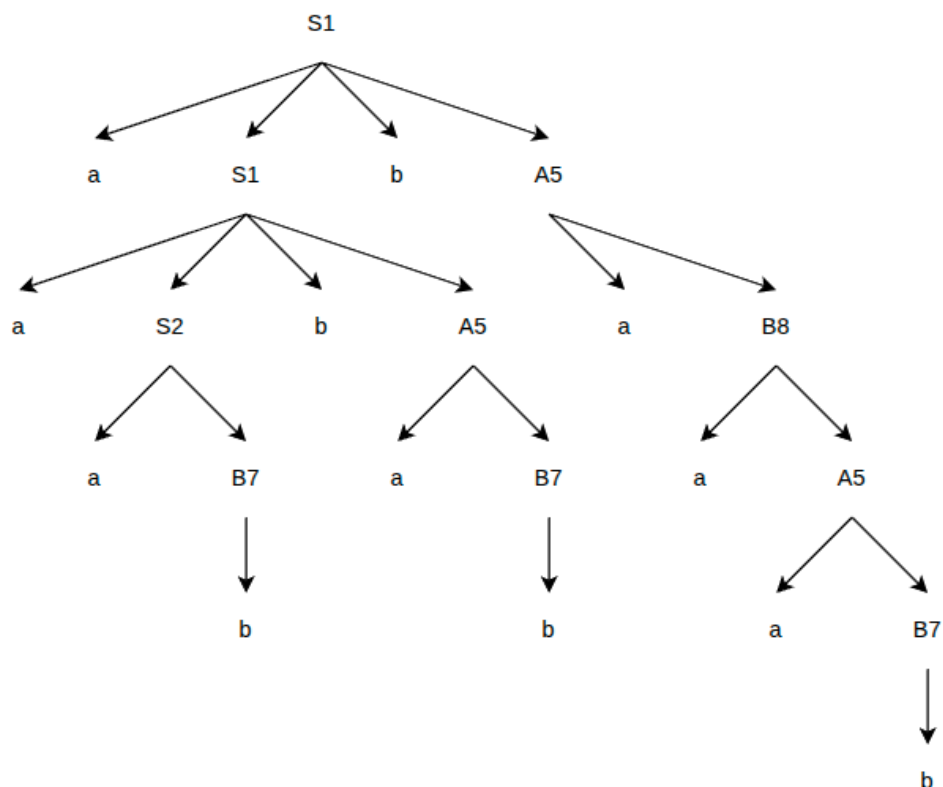
Левый вывод 1:

$S_1 \Rightarrow aS_1bA \Rightarrow aaS_2bAbA \Rightarrow aaaB_7bAbA \Rightarrow aaabbA_5bA \Rightarrow aaabbaB_7bA \Rightarrow aaabbabbA_5 \Rightarrow aaabbabbaB_8 \Rightarrow aaabbabbaaA_5 \Rightarrow aaabbabbaaaB_7 \Rightarrow aaabbabbaaab$

Терминальная цепочка: *aaabbabbaaab*

Последовательность правил: 1 1 2 7 5 7 5 8 5 7

Дерево вывода:



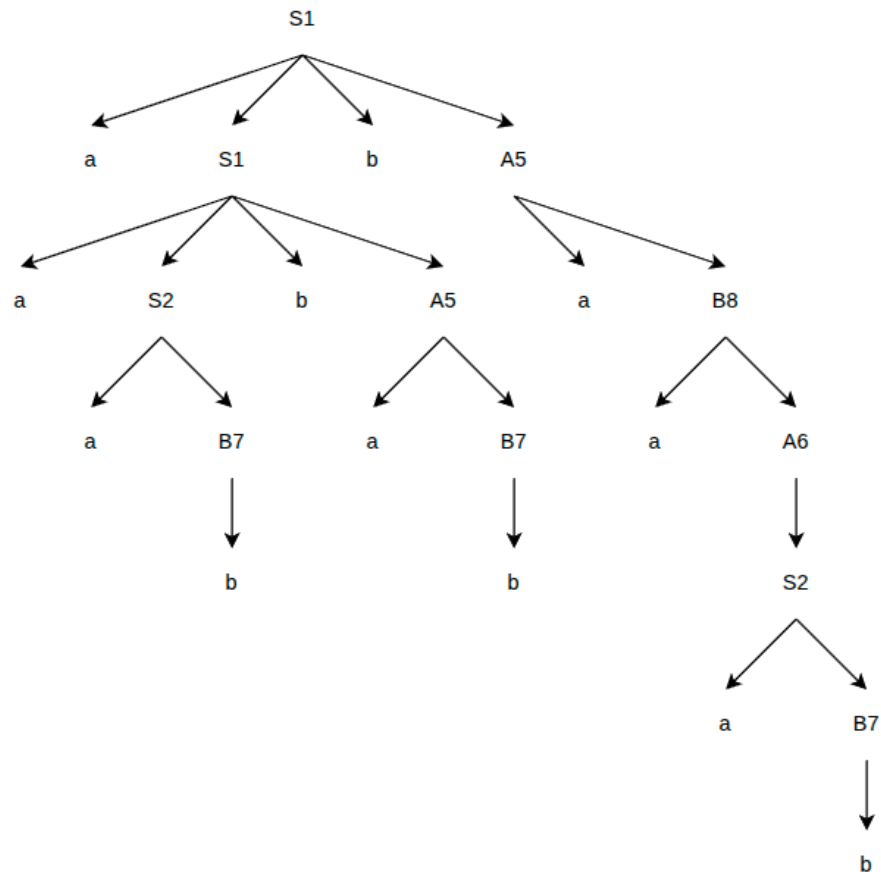
Левый вывод 2:

$S_1 \Rightarrow a S_1 b A \Rightarrow aa S_2 b A b A \Rightarrow aaa B_7 b A b A \Rightarrow aaabb A_5 b A \Rightarrow aaabba B_7 b A \Rightarrow aaabbabb A_5 \Rightarrow aaabbabba B_8 \Rightarrow aaabbabbaa A_6 \Rightarrow aabbabbaa S_2 \Rightarrow aaabbabbaaa B_7 \Rightarrow aaabbabbaaab$

Терминальная цепочка: *aaabbabbaaab*

Последовательность правил: 1 1 2 7 5 7 5 8 6 2 7

Дерево вывода:



2. Написать программу, которая определяет, можно ли применить заданную последовательность правил при левом выводе терминальной цепочки в заданной КС-грамматике, формирует левый вывод и линейную скобочную форму дерева вывода.

Обработать программой последовательности правил, полученные в п.1.

Примечание. Если к нетерминалу *A* в процессе вывода применяется правило с номером *n*, то в выводе и в линейной скобочной форме дерева вывода после нетерминала *A* должен быть символ с кодом *n*.

Исходный код:

```
#include <bits/stdc++.h>
using namespace std;

/* ----- 1. Структура правила и инициализация грамматики ----- */
struct Rule {
    string lhs; // левые члены (непустой символ)
    string rhs; // правые члены
    int num; // номер правила
};

static const vector<Rule> rules = {
    {"S", "aSbA", 1},
    {"S", "aB", 2},
}
```

```

{"S", "A", 3},
{"A", "aAbS", 4},
{"A", "aB", 5},
{"A", "S", 6},
{"B", "b", 7},
{"B", "aA", 8}
};

/* ----- 2. Узел дерева вывода ----- */
struct Node {
    string symbol;      // символ
    bool terminal;      // true – терминал
    int ruleNum = -1;    // номер правила, которым он был развернут
    vector<Node*> children; // потомки

    Node(const string& s, bool t = true) : symbol(s), terminal(t) {}
};

/* ----- 3. Поиск правила по номеру ----- */
const Rule* findRule(int num) {
    for (const auto& r : rules)
        if (r.num == num) return &r;
    return nullptr;
}

/* ----- 4. Функция вывода лексической скобочной формы ----- */
void printNode(Node* n, string &out) {
    if (n->terminal) {
        out += n->symbol;
        return;
    }
    out += '(' + n->symbol + to_string(n->ruleNum);
    for (Node* c : n->children) printNode(c, out);
    out += ')';
}

/* ----- 5. Главная функция ----- */
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    /* ----- ввод последовательности номеров правил ----- */
    vector<int> seqRules;
    int x;
    while (cin >> x) seqRules.push_back(x);

    /* ----- инициализация дерева и списка текущих нод ----- */
    Node* root = new Node("S", false);
    vector<Node*> curList = {root};

    bool ok = true;

    for (int ruleNum : seqRules) {
        const Rule* r = findRule(ruleNum);
        if (!r) { ok = false; break; }

        /* поиск левого самого первого нерегулярного символа */
        size_t pos = 0;
        while (pos < curList.size() && curList[pos]->terminal) ++pos;
        if (pos == curList.size()) { ok = false; break; }

        Node* target = curList[pos];
        if (target->symbol != r->lhs) { ok = false; break; }
    }
}

```

```

/* развернуть правило */
target->ruleNum = r->num;
target->terminal = false;
target->children.clear();

/* создаём потомков и обновляем список текущих нод */
vector<Node*> newNodes;
for (char ch : r->rhs) {
    if (isupper(ch)) { // нетерминал
        Node* child = new Node(string(1, ch), false);
        target->children.push_back(child);
        newNodes.push_back(child);
    } else { // терминал
        Node* child = new Node(string(1, ch), true);
        target->children.push_back(child);
        newNodes.push_back(child);
    }
}

curList.erase(curList.begin() + pos, curList.begin() + pos + 1);
curList.insert(curList.begin() + pos, newNodes.begin(), newNodes.end());
}

/* ----- проверка, что все стали терминалами ----- */
for (Node* n : curList)
    if (!n->terminal) ok = false;

/* ----- вывод результата ----- */
if (!ok) {
    cout << "Невозможно применить заданную последовательность правил.\n";
    return 0;
}

/* строка левого вывода */
string derivation;
for (Node* n : curList) derivation += n->symbol;
cout << "Левый вывод: " << derivation << "\n";

/* линейная скобочная форма */
string linear;
printNode(root, linear);
cout << "Линейная скобочная форма: " << linear << "\n";

return 0;
}

```

Результат выполнения программы, для p1:

```

1 1 2 7 5 7 5 8 5 7
q
Возможно применить заданную последовательность правил.
Левый вывод: aaabbabbaaab
Линейная скобочная форма: (S1a(S1a(S2a(B7b))b(A5a(B7b)))b(A5a(B8a(A5a(B7b))))

```

Результат выполнения программы, для p2:

```

1 1 2 7 5 7 5 8 6 2 7
q
Возможно применить заданную последовательность правил.
Левый вывод: aaabbabbaaab
Линейная скобочная форма: (S1a(S1a(S2a(B7b))b(A5a(B7b)))b(A5a(B8a(A6(S2a(B7b))))

```

3. Найти последовательность правил  $p$ ,  $|p| > 10$ , которую можно применить при произвольном выводе терминальной цепочки, но нельзя применить при левом или правом выводе в заданной КС-грамматике (см. варианты заданий).

Записать вывод  $v$ , в процессе которого применяется последовательность правил  $p$ . Построить дерево вывода.

Записать левый и правый выводы, эквивалентные выводу  $v$ .

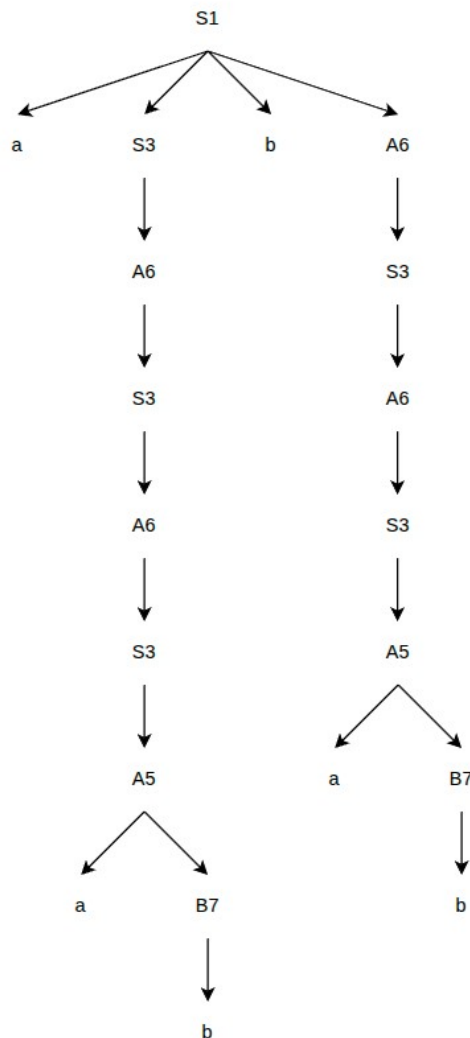
### Произвольный вывод:

$S_1 \Rightarrow aS_3bA \Rightarrow aAbA_6 \Rightarrow aA_6bS \Rightarrow aSbS_3 \Rightarrow aS_3bA \Rightarrow aAb a_6 \Rightarrow aA_6bS \Rightarrow aSbS_3 \Rightarrow aS_3bA \Rightarrow aAbA_5 \Rightarrow aA_5baB \Rightarrow aaBbaB_7 \Rightarrow aaB_7bab \Rightarrow aabbab$

Последовательность правил  $p$ : 1 3 6 6 3 3 6 6 3 3 5 5 7 7

Вывод  $v$ :  $aabbab$

Дерево вывода:



### Левый вывод:

$S_1 \Rightarrow aS_3bA \Rightarrow aA_6bA \Rightarrow aS_3bA \Rightarrow aA_6bA \Rightarrow aS_3bA \Rightarrow aA_5bA \Rightarrow aaB_7bA \Rightarrow aabbA_6 \Rightarrow aabbS_3 \Rightarrow aabbA_6 \Rightarrow aabbS_3 \Rightarrow aabbA_5 \Rightarrow aabbaB_7 \Rightarrow aabbab$

Последовательность правил: 1 3 6 3 6 3 5 7 6 3 6 3 5 7

Вывод:  $aabbab$

### Правый вывод:

$S_1 \Rightarrow aSb A_6 \Rightarrow aSb S_3 \Rightarrow aSb A_6 \Rightarrow aSb S_3 \Rightarrow aSb A_5 \Rightarrow aSb A_7 \Rightarrow aS_3 bab \Rightarrow a A_6 bab \Rightarrow a S_3 bab \Rightarrow a A_6 bab \Rightarrow a S_3 bab \Rightarrow a A_5 bab \Rightarrow aa B_7 bab \Rightarrow aabbab$

Последовательность правил: 1 6 3 6 3 5 7 3 6 3 6 3 5 7

Вывод: *aabbab*

Видим, что левый и правый выводы эквивалентный выводу  $v$  (Произвольный вывод), также последовательность правил  $p$  можно применить только в произвольном выводе, для левого и правого выводов ее применить невозможно.

4. Написать программу, которая определяет, можно ли применить заданную последовательность правил  $p$  при выводе терминальной цепочки в заданной КС-грамматике и формирует линейную скобочную форму дерева вывода. Если последовательность правил  $p$  можно применить при выводе  $v$  терминальной цепочки, то программа должна вывести последовательность правил, применяемую при левом выводе, эквивалентном выводу  $v$ .

Обработать программой последовательность правил, найденную в п.3.

Примечание. Если к нетерминалу  $A$  в процессе вывода применяется правило с номером  $n$ , то в линейной скобочной форме дерева вывода после нетерминала  $A$  должен быть символ с кодом  $n$ .

### Исходный код:

```
#include <bits/stdc++.h>
using namespace std;

// Грамматика:
// 1.  $S \rightarrow a S b A$ 
// 2.  $S \rightarrow a B$ 
// 3.  $S \rightarrow A$ 
// 4.  $A \rightarrow a A b S$ 
// 5.  $A \rightarrow a B$ 
// 6.  $A \rightarrow S$ 
// 7.  $B \rightarrow b$ 
// 8.  $B \rightarrow a A$ 

struct Node
{
    char sym; // 'S', 'A', 'B' для нетерминалов или 'a', 'b' для терминалов
    int rule; // номер правила, использованного для раскрытия узла, -1 если не раскрыт (лист)
    vector<shared_ptr<Node>> ch;
    Node(char s = 0) : sym(s), rule(-1) {}
};

// Проверка: символ — нетерминал (заглавная буква)
bool is_nonterm(char c) { return c >= 'A' && c <= 'Z'; }
bool is_terminal(char c) { return !is_nonterm(c); }

// Отображение правил: номер_правила -> пара(lhs, rhs_string)
map<int, pair<char, string>> productions;

shared_ptr<Node> deep_copy(const shared_ptr<Node> &root)
```

```

{
    if (!root)
        return nullptr;
    auto n = make_shared<Node>(root->sym);
    n->rule = root->rule;
    for (auto &c : root->ch)
        n->ch.push_back(deep_copy(c));
    return n;
}

// Собрать листья (слева направо) в вектор
void collect_leaves(const shared_ptr<Node> &node, vector<shared_ptr<Node>> &leaves)
{
    if (node->ch.empty())
    {
        leaves.push_back(node);
    }
    else
    {
        for (auto &c : node->ch)
            collect_leaves(c, leaves);
    }
}

// Раскрыть лист 'leaf' с помощью продукции (lhs -> rhs) и установить номер правила
void expand_leaf(shared_ptr<Node> leaf, int rule_num, const string &rhs)
{
    leaf->rule = rule_num;
    leaf->ch.clear();
    for (char c : rhs)
    {
        auto child = make_shared<Node>(c);
        leaf->ch.push_back(child);
    }
}

// Проверить, что все листья — терминалы
bool leaves_all_terminals(const shared_ptr<Node> &root)
{
    vector<shared_ptr<Node>> leaves;
    collect_leaves(root, leaves);
    for (auto &l : leaves)
        if (is_nonterm(l->sym))
            return false;
    return true;
}

// Построить линейную скобочную форму:
string linear_form(const shared_ptr<Node> &node)
{
    if (node->ch.empty())
    {
        // лист: терминал или нераскрытый нетерминал
        string s(1, node->sym);
        return s;
    }
    else
    {
        string res;
        res.push_back(node->sym);
        if (node->rule != -1)
            res += to_string(node->rule);
    }
}

```



```

res += "(";
bool first = true;
for (auto &c : node->ch)
{
    if (!first)
        res += ",";
    res += linear_form(c);
    first = false;
}
res += ")";
return res;
}
}

// Собрать последовательность правил левого вывода из дерева:
// посещаем узел: если раскрыт (rule!=-1), добавляем правило, затем рекурсивно обходим детей слева направо
void collect_leftmost_seq(const shared_ptr<Node> &node, vector<int> &seq)
{
    if (node->ch.empty())
        return;
    if (node->rule != -1)
        seq.push_back(node->rule);
    for (auto &c : node->ch)
        collect_leftmost_seq(c, seq);
}

// Поиск в глубину/бэктрекинг для применения заданной последовательности правил
// возвращает true и устанавливает 'result_tree' при успехе
bool dfs_try(const shared_ptr<Node> &tree, const vector<int> &seq, int idx, shared_ptr<Node> &result_tree)
{
    if (idx == (int)seq.size())
    {
        // все правила применены: проверяем, что все листья терминальные
        if (leaves_all_terminals(tree))
        {
            result_tree = tree;
            return true;
        }
        else
            return false;
    }
    int rule_num = seq[idx];
    if (productions.find(rule_num) == productions.end())
        return false;
    char lhs = productions[rule_num].first;
    string rhs = productions[rule_num].second;

    // собрать листья текущего дерева (слева направо), попытаться раскрыть любой лист с символом lhs
    vector<shared_ptr<Node>> leaves;
    collect_leaves(tree, leaves);

    bool anyAttempt = false;
    for (size_t i = 0; i < leaves.size(); ++i)
    {
        if (leaves[i]->sym == lhs)
        {
            anyAttempt = true;
            auto newtree = deep_copy(tree);
            // найти i-й лист в копии:
            vector<shared_ptr<Node>> newleaves;
            collect_leaves(newtree, newleaves);
            // раскрыть

```

```

    expand_leaf(newleaves[i], rule_num, rhs);
    // рекурсия
    if (dfs_try(newtree, seq, idx + 1, result_tree))
        return true;
}
}
// Если подходящего LHS не найдено -> тупик
if (!anyAttempt)
    return false;
return false;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // Инициализация productions грамматики
    productions[1] = {'S', string("aSbA")};
    productions[2] = {'S', string("aB")};
    productions[3] = {'S', string("A")};
    productions[4] = {'A', string("aAbS")};
    productions[5] = {'A', string("aB")};
    productions[6] = {'A', string("S")};
    productions[7] = {'B', string("b")};
    productions[8] = {'B', string("aA")};

    string line;
    if (!getline(cin, line))
        return 0;
    // обрезать пробелы
    while (!line.empty() && isspace(line.back()))
        line.pop_back();
    while (!line.empty() && isspace(line.front()))
        line.erase(line.begin());

    vector<int> seq;
    if (line.find(' ') != string::npos)
    {
        // разделить по пробелам (поддержка многозначных чисел)
        stringstream ss(line);
        int x;
        while (ss >> x)
            seq.push_back(x);
    }
    else
    {
        // нет пробелов -> считаем каждый символ цифрой (например, "12332")
        for (char c : line)
        {
            if (isdigit(c))
            {
                seq.push_back(c - '0');
            }
            else if (!isspace(c))
            {
                cerr << "Неожиданный символ во входе: " << c << "\n";
                return 0;
            }
        }
    }
}

```

```

if (seq.empty())
{
    cout << "Пустая последовательность правил. Выход.\n";
    return 0;
}

// стартовое дерево: один узел S
auto start = make_shared<Node>('S');

shared_ptr<Node> result_tree = nullptr;
bool ok = dfs_try(start, seq, 0, result_tree);

if (!ok)
{
    cout << "Последовательность правил **НЕ** может быть применена для получения терминальной цепочки в заданной грамматике.\n";
    return 0;
}

// Построить линейную скобочную форму
string lin = linear_form(result_tree);

// Построить терминальную строку v (конкатенация листьев)
vector<shared_ptr<Node>> leaves;
collect_leaves(result_tree, leaves);
string v;
for (auto &l : leaves)
    v.push_back(l->sym);

// Построить последовательность правил левого вывода из дерева
vector<int> left_seq;
collect_leftmost_seq(result_tree, left_seq);

cout << "Последовательность правил может быть применена.\n";
cout << "\nПолученная терминальная цепочка v = \"" << v << "\"\n";
cout << "\nЛинейная скобочная форма дерева вывода:\n";
cout << lin << "\n";
cout << "\nПоследовательность правил при левом выводе, эквивалентном выводу v:\n";
for (size_t i = 0; i < left_seq.size(); ++i)
{
    if (i)
        cout << " ";
    cout << left_seq[i];
}
cout << "\n";
return 0;
}

```

Результат выполнения программы:

```

13663366335577
Последовательность правил может быть применена.

Полученная терминальная цепочка v = "aabbab"

Линейная скобочная форма дерева вывода:
S1(aS3(A6(S3(A6(S3(A5(aB7(b)))))))bA6(S3(A6(S3(A5(aB7(b)))))))

Последовательность правил при левом выводе, эквивалентном выводу v:
1 3 6 3 6 3 5 7 6 3 6 3 5 7

```

Вывод: в ходе выполнения л.р. я изучил основные понятия теории формальных языков и грамматик.