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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

по дисциплине: Теория автоматов и формальных языков
тема: «Нисходящая обработка контекстно-свободных языков»

Выполнил: ст. группы ПВ-223
Игнатьев Артур Олегович
Проверил:
Рязанов Юрий Дмитриевич

Белгород 2024 г.

Цель работы: изучить и научиться применять нисходящие методы обработки формальных языков.

Вариант 3

1. S→OS
2. S→O
3. O→a[S]
4. O→a[S][S]
5. O→ a=E
6. E→E+T
7. E→T
8. T→T*P
9. T→P
10. P→(E)
11. P→a

Задание

1. Преобразовать исходную КС-грамматику в LL(1)-грамматику (см. варианты заданий).
2. Определить множества ПЕРВЫХ для каждого символа LL(1)-грамматики.
3. Определить множества СЛЕДУЮЩИХ для каждого символа LL(1)-грамматики.
4. Определить множество ВЫБОРА для каждого правила LL(1)-грамматики.
5. Написать программу-распознаватель методом рекурсивного спуска. Программа должна выводить последовательность номеров правил, применяемых при левом выводе обрабатываемой цепочки.

6. Сформировать наборы тестовых данных. Тестовые данные должны содержать цепочки, принадлежащие языку, заданному грамматикой, (допустимые цепочки) и цепочки, не принадлежащие языку. Для каждой допустимой цепочки построить дерево вывода и левый вывод.

Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз.

7. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.

8. Построить нисходящий МП-распознаватель по LL(1)-грамматике.

9. Написать программу-распознаватель, реализующую построенный нисходящий МП-распознаватель. Программа должна выводить на каждом шаге номер применяемого правила и промежуточную цепочку левого вывода.

10. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.

Задание 1. Преобразовать исходную КС-грамматику в LL(1)-грамматику (см. варианты заданий).

1. S→OS
2. S→O
3. O→a[S]
4. O→a[S][S]
5. O→ a=E
6. E→E+T
7. E→T
8. T→T*P
9. T→P
10. P→(E)
11. P→a

Грамматика не является LL(1). Видим пересечение множеств выбора для нетерминала O: в правилах 3, 4 и 5.

Грамматика имеет левую рекурсию. Устраним ее:

1. S → OS
2. S → O
3. O → a[S]
4. O → a[S][S]
5. O → a=E
6. E → TE’
7. E’ → +TE’
8. E’ → ε
9. T → PT’
10. T’ → *PT’
11. T’ → ε

12. P → (E)

13. P → a

Проводим левую факторизацию и замену края пока возможно:

1)

1. S → O N1

2. N1 → S

3. N1 → ε

3. O → a[S]

4. O → a[S][S]

5. O → a=E

6. E → TE’

7. E’ → +TE’

8. E’ → ε

9. T → PT’

10. T’ → *PT’

11. T’ → ε

12. P → (E)

13. P → a

2)

1. S → O N1

2. N1 → S

3. N1 → ε

4. O → aN2

5. N2 → [S]

6. N2 → [S][S]

7. N2 → =E

8. E → TE’

9. E’ → +TE’

10. E’ → ε

11. T → PT’

12. T’ → *PT’

13. T’ → ε

14. P → €

15. P → a

3)

1. S → aN2N1

2. N1 → aN2N1

3. N1 → ε

4. O → aN2

5. N2 → [S]

6. N2 → [S][S]

7. N2 → =E

8. E → TE’

9. E’ → +TE’

10. E’ → ε

11. T → PT’

12. T’ → *PT’

13. T’ → ε

14. P → (E)

15. P → a

4)

1. S → aN2N1

2. N1 → aN2N1

3. N1 → ε

4. O → aN2

5. N2 → [S] N3

6. N3 → [S]

7. N3 → ε

8. N2 → =E

9. E → TE’

10. E’ → +TE’

11. E’ → ε

12. T → PT’

13. T’ → *PT’

14. T’ → ε

15. P → (E)

16. P → a

5)

1. S → aN2N1

2. N1 → aN2N1

3. N1 → ε

4. O → aN2

5. N2 → [S] N3

6. N2 → =E

7. N3 → [S]

8. N3 → ε

9. E → (E)T’E’

10. E → aT’E’

11. E’ → +TE’

12. E’ → ε

13. T → (E)T’

14. T → aT’

15. T’ → *PT’

16. T’ → ε

16. P → (E)

17. P → a

Устраним правило 4, как недостижимое

1. S → aN2N1

2. N1 → aN2N1

3. N1 → ε

4. N2 → [S] N3

5. N2 → =E

6. N3 → [S]

7. N3 → ε

8. E → (E)T’E’

9. E → aT’E’

10. E’ → +TE’

11. E’ → ε

12. T → (E)T’

13. T → aT’

14. T’ → *PT’

15. T’ → ε

16. P → (E)

17. P → a

Получили предположительную LL(1) грамматику и проведем проверку соответствия.

Задание 2. Определить множества ПЕРВЫХ для каждого символа LL(1)-грамматики.

Определим множества ПЕРВЫХ:

1. S → aN2N1 {a}
2. N1 → aN2N1
3. N1 → ε {a, ε}
4. N2 → [S] N3
5. N2 → =E {[, =}
6. N3 → [S]
7. N3 → ε {[, ε}
8. E → (E)T’E’
9. E → aT’E’ {(, a}
10. E’ → +TE’
11. E’ → ε {+, ε}
12. T → (E)T’
13. T → aT’ {(, a}
14. T’ → *PT’
15. T’ → ε {*, ε }
16. P → (E)
17. P → a {(, a}

Задание 3. Определить множества СЛЕДУЮЩИХ для каждого символа LL(1)-грамматики.

Определим множества СЛЕДУЮЩИХ:

1. S → aN2N1 {┤,]}
2. N1 → aN2N1
3. N1 → ε {┤,]}
4. N2 → [S] N3
5. N2 → =E {a, ┤,]}
6. N3 → [S]
7. N3 → ε {a, ┤,]}
8. E → (E)T’E’
9. E → aT’E’ {), a, ┤,]}
10. E’ → +TE’
11. E’ → ε {), a, ┤,]}
12. T → (E)T’
13. T → aT’ {+,), a, ┤,]}
14. T’ → *PT’
15. T’ → ε {+,), a, ┤,]}
16. P → (E)
17. P → a {*, +,), a, ┤,]}

Задание 4. Определить множество ВЫБОРА для каждого правила LL(1)-грамматики.

Определим множества ВЫБОРА:

1. S → aN2N1 {a}
2. N1 → aN2N1 {a}
3. N1 → ε {┤,]}
4. N2 → [S] N3 {[}
5. N2 → =E {=}
6. N3 → [S] {[}
7. N3 → ε {a, ┤,]}
8. E → (E)T’E’ {(}
9. E → aT’E’ {a}
10. E’ → +TE’ {+}
11. E’ → ε {), a, ┤,]}
12. T → (E)T’ {(}
13. T → aT’ {a}
14. T’ → *PT’ {*}
15. T’ → ε {+,), a, ┤,]}
16. P → (E) {(}
17. P → a {a}

Из полученных множеств ВЫБОРА сделаем вывод, что в первом задании получена LL(1) грамматика, так как нет пересечений множеств ВЫБОРА.

Задание 5. Написать программу-распознаватель методом рекурсивного спуска. Программа должна выводить последовательность номеров правил, применяемых при левом выводе обрабатываемой цепочки.

Программа написана на языке Java

```
class LL1Parser {
    private static class InvalidStringError extends Exception {
        public InvalidStringError() {
            super("Недопустимая строка");
        }
    }

    private String terminalIdentification(String data, char terminal) throws
InvalidStringError {
        if (data.charAt(0) == terminal) {
            return data.substring(1);
        } else {
            throw new InvalidStringError();
        }
    }

    private String S(String data) throws InvalidStringError {
        if (data.charAt(0) == 'a') {
            System.out.println("1. S -> aN2N1");
            data = terminalIdentification(data, 'a');
            data = N2(data);
            data = N1(data);
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String N1(String data) throws InvalidStringError {
        if (data.charAt(0) == 'a') {
            System.out.println("2. N1 -> aN2N1");
            data = terminalIdentification(data, 'a');
            data = N2(data);
            data = N1(data);
        } else if (data.charAt(0) == '┤' || data.charAt(0) == ']') {
            System.out.println("3. N1 -> ε");
            // Пропускаем, возвращаем строку как есть
        } else {
            throw new InvalidStringError();
        }
```

```
        return data;
    }

    private String N2(String data) throws InvalidStringError {
        if (data.charAt(0) == '[') {
            System.out.println("4. N2 -> [S] N3");
            data = terminalIdentification(data, '[');
            data = S(data);
            data = terminalIdentification(data, ']');
            data = N3(data);
        } else if (data.charAt(0) == '=') {
            System.out.println("5. N2 -> =E");
            data = terminalIdentification(data, '=');
            data = E(data);
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String N3(String data) throws InvalidStringError {
        if (data.charAt(0) == '[') {
            System.out.println("6. N3 -> [S]");
            data = terminalIdentification(data, '[');
            data = S(data);
            data = terminalIdentification(data, ']');
        } else if (data.charAt(0) == 'a' || data.charAt(0) == '┤' || data.cha-
rAt(0) == ']') {
            System.out.println("7. N3 -> ε");
            // Пропускаем, возвращаем строку как есть
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String E(String data) throws InvalidStringError {
        if (data.charAt(0) == '(') {
            System.out.println("8. E -> (E)T'E'");
            data = terminalIdentification(data, '(');
            data = E(data);
            data = terminalIdentification(data, ')');
            data = T_prime(data);
            data = E_prime(data);
        } else if (data.charAt(0) == 'a') {
            System.out.println("9. E -> aT'E'");
            data = terminalIdentification(data, 'a');
            data = T_prime(data);
```

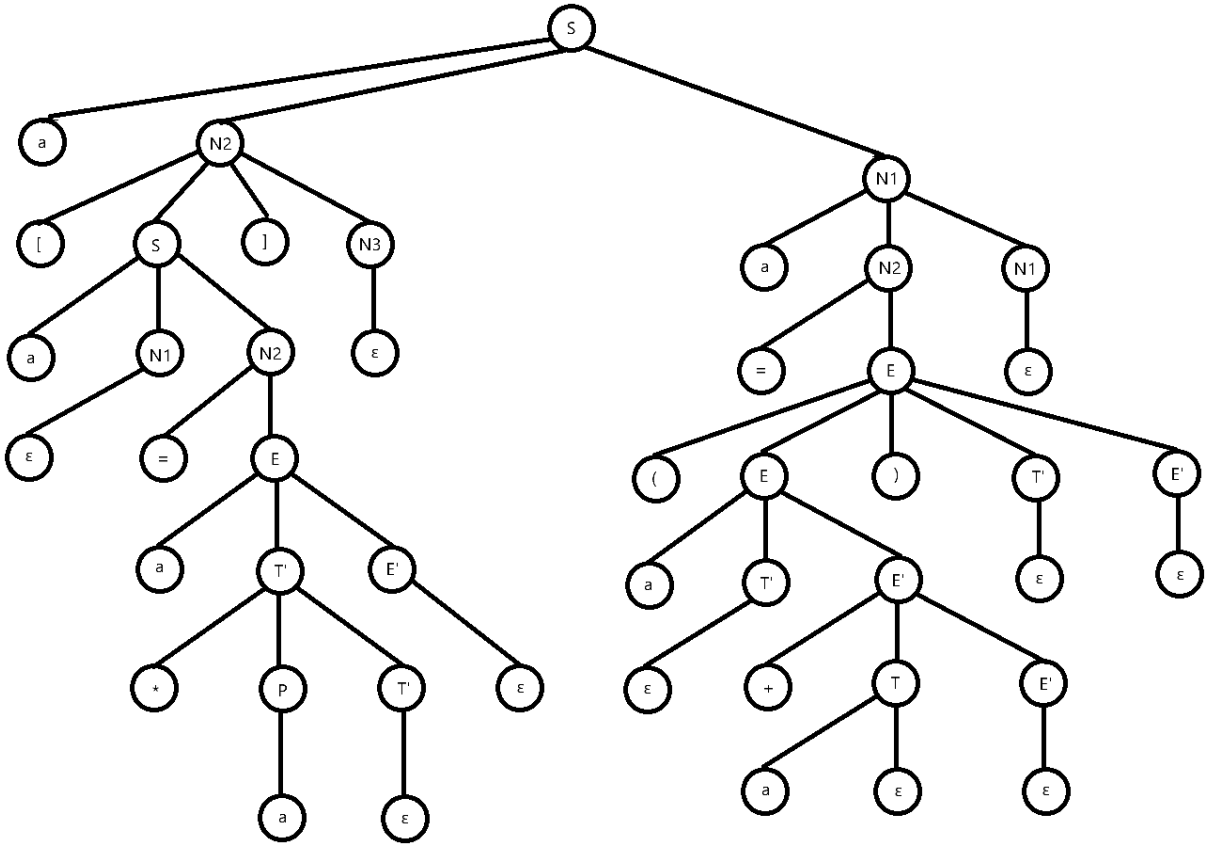
```
            data = E_prime(data);
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String E_prime(String data) throws InvalidStringError {
        if (data.charAt(0) == '+') {
            System.out.println("10. E' -> +TE'");
            data = terminalIdentification(data, '+');
            data = T(data);
            data = E_prime(data);
        } else if (data.charAt(0) == ')' || data.charAt(0) == 'a' || data.cha-
rAt(0) == '┤' || data.charAt(0) == ']') {
            System.out.println("11. E' -> ε");
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String T(String data) throws InvalidStringError {
        if (data.charAt(0) == '(') {
            System.out.println("12. T -> (E)T'");
            data = terminalIdentification(data, '(');
            data = E(data);
            data = terminalIdentification(data, ')');
            data = T_prime(data);
        } else if (data.charAt(0) == 'a') {
            System.out.println("13. T -> aT'");
            data = terminalIdentification(data, 'a');
            data = T_prime(data);
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String T_prime(String data) throws InvalidStringError {
        if (data.charAt(0) == '*') {
            System.out.println("14. T' -> *PT'");
            data = terminalIdentification(data, '*');
            data = P(data);
            data = T_prime(data);
        } else if (data.charAt(0) == '+' || data.charAt(0) == ')' || data.cha-
rAt(0) == 'a' || data.charAt(0) == '┤' || data.charAt(0) == ']') {
            System.out.println("15. T' -> ε");
```

```
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    private String P(String data) throws InvalidStringError {
        if (data.charAt(0) == '(') {
            System.out.println("16. P -> (E)");
            data = terminalIdentification(data, '(');
            data = E(data);
            data = terminalIdentification(data, ')');
        } else if (data.charAt(0) == 'a') {
            System.out.println("17. P -> a");
            data = terminalIdentification(data, 'a');
        } else {
            throw new InvalidStringError();
        }
        return data;
    }

    public boolean LL1(String data) {
        System.out.printf("Путь применяемых правил для %s:%n", data);
        data += "┤";
        try {
            String result = S(data);
            if (result.equals("┤")) {
                System.out.println("Цепочка допустима");
                System.out.println("-".repeat(30));
                return true;
            } else {
                System.out.println("Цепочка недопустима");
                System.out.println("-".repeat(30));
                return false;
            }
        } catch (InvalidStringError e) {
            System.out.println("Цепочка недопустима");
            System.out.println("-".repeat(30));
            return false;
        }
    }

    public static void main(String[] args) {
        LL1Parser parser = new LL1Parser();
        parser.LL1("a[a=a*a]a=(a+a)");
        parser.LL1("a[a+a]a");
        parser.LL1("bscsds");
    }}
```

Задание 6. Сформировать наборы тестовых данных. Тестовые данные должны содержать цепочки, принадлежащие языку, заданному грамматикой, (допустимые цепочки) и цепочки, не принадлежащие языку. Для каждой допустимой цепочки построить дерево вывода и левый вывод. Каждое правило грамматики должно использоваться в выводах допустимых цепочек хотя бы один раз.

Проверка 1: a[a=a*a]a=(a+a)

Дерево вывода:



Левый вывод:

$$S \Rightarrow aN2N1 \Rightarrow a[S]N3N1 \Rightarrow a[aN1N2]N3N1 \Rightarrow a[aN2]N3N1$$
$$\Rightarrow a[a = E]N3N1 \Rightarrow a[a = aT'E']N3N1$$
$$\Rightarrow a[a = a * PT'E']N3N1 \Rightarrow a[a = a * aT'E']N3N1 \Rightarrow a[a$$
$$= a * aE']N3N1 \Rightarrow\ a[a = a * a]N3N1 \Rightarrow a[a = a * a]N1$$
$$\Rightarrow a[a = a * a]aN2N1 \Rightarrow a[a = a * a]a = EN1 \Rightarrow\ a[a$$
$$= a * a]a = (E)T'E'N1 \Rightarrow a[a = a * a]a = (aT'E')T'E'N1$$
$$\Rightarrow\ a[a = a * a]a = (aE')T'E'N1 \Rightarrow\ a[a = a * a]a$$
$$= (a + TE')T'E'N1 \Rightarrow a[a = a * a]a = (a + aT'E')T'E'N1$$
$$\Rightarrow\ a[a = a * a]a = (a + aE')T'E'N1 \Rightarrow\ a[a = a * a]a$$
$$= (a + a)T'E'N1 \Rightarrow\ a[a = a * a]a = (a + a)E'N1 \Rightarrow\ a[a$$
$$= a * a]a = (a + a)N1 \Rightarrow\ a[a = a * a]a = (a + a)$$

Проверка 2: a[a+a]a

Ожидаемый результат: недопустимая цепочка

Проверка 3: bscsds (случайные символы)

Ожидаемый результат: недопустимая цепочка

Задание 7. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.

Результат работы программы:

```
Путь применяемых правил для a[a=a*a]a=(a+a):
1. S -> aN2N1
4. N2 -> [S] N3
1. S -> aN2N1
5. N2 -> =E
9. E -> aT'E'
14. T' -> *PT'
17. P -> a
15. T' -> ε
11. E' -> ε
3. N1 -> ε
7. N3 -> ε
2. N1 -> aN2N1
5. N2 -> =E
8. E -> (E)T'E'
9. E -> aT'E'
15. T' -> ε
10. E' -> +TE'
13. T -> aT'
15. T' -> ε
11. E' -> ε
15. T' -> ε
11. E' -> ε
3. N1 -> ε
Цепочка допустима
------------------------------
Путь применяемых правил для a[a+a]a:
1. S -> aN2N1
4. N2 -> [S] N3
1. S -> aN2N1
Цепочка недопустима
------------------------------
Путь применяемых правил для bscsds:
Цепочка недопустима
------------------------------
```

Задание 8. Построить нисходящий МП-распознаватель по LL(1)-грамматике.

МП-распознаватель:

	a	**[**	**]**	**=**	**(**	**)**	**+**	*****	┤
S	#1								
N1	#2		#3						#3
N2		#4		#5					
N3	#7	#6	#7						#7
E	#9				#8				
E’	#11		#11			#11	#10		#11
T	#13				#12				
T’	#15		#15			#15	#15	#14	#15
P	#17				#16				
}			Выт сд						
)						Выт сд			
Δ									доп

1#: ЗАМЕНИТЬ (N1N2), сдвиг

2#: ЗАМЕНИТЬ (N1N2), сдвиг

3#: ВЫТОЛКНУТЬ ДЕРЖАТЬ

4#: ЗАМЕНИТЬ (N3[S), сдвиг

5#: ЗАМЕНИТЬ (E), сдвиг

6#: ЗАМЕНИТЬ ([S), сдвиг

7#: ВЫТОЛКНУТЬ ДЕРЖАТЬ

8#: ЗАМЕНИТЬ (E’T’(E), сдвиг

9#: ЗАМЕНИТЬ (E’T’), сдвиг

10#: ЗАМЕНИТЬ (E’T), сдвиг

11#: ВЫТОЛКНУТЬ ДЕРЖАТЬ

12#: ЗАМЕНИТЬ (Т’ (Е), сдвиг

13#: ЗАМЕНИТЬ (Т’), сдвиг

14#: ЗАМЕНИТЬ (Т’Р), сдвиг

15#: ВЫТОЛКНУТЬ ДЕРЖАТЬ

16#: ЗАМЕНИТЬ ((Е), сдвиг

17#: ВЫТОЛКНУТЬ, сдвиг

Задание 9. Написать программу-распознаватель, реализующую построенный нисходящий МП-распознаватель. Программа должна выводить на каждом шаге номер применяемого правила и промежуточную цепочку левого вывода.

Были переименованы нетерминалы N1, N2, N3, E’ и T’ на G, F, Q, V и L, для большего удобства реализации стека.

Программа написана на языке Java:

```
public class PushdownParser {
    private static class InvalidStringError extends RuntimeException {
        public InvalidStringError() {
            super("Invalid string detected");
        }
    }

    private String string;
    private String originString;
    private String stack;
    private int shouldIter;

    public PushdownParser(String string) {
        this.string = string;
        this.originString = string;
        this.stack = "SΔ";
        this.shouldIter = -1;
    }

    public boolean parse() {
        System.out.printf("Путь применяемых правил для %s:%n", string);
        try {
            string += "┤";
            while (shouldIter == -1) {
                char m = stack.charAt(0);
                char x = string.charAt(0);
                if (m == 'Δ' && string.equals("┤")) {
                    shouldIter = string.equals("┤") ? 1 : -1;
                    if (shouldIter == -1) {
                        throw new InvalidStringError();
                    }
                } else {
                    switch (m) {
```

```
                        case 'S' -> processRule(S(string, stack, ori-
ginString));
                        case 'G' -> processRule(G(string, stack, ori-
ginString));
                        case 'F' -> processRule(F(string, stack, ori-
ginString));
                        case 'Q' -> processRule(Q(string, stack, ori-
ginString));
                        case 'E' -> processRule(E(string, stack, ori-
ginString));
                        case 'V' -> processRule(V(string, stack, ori-
ginString));
                        case 'T' -> processRule(T(string, stack, ori-
ginString));
                        case 'L' -> processRule(L(string, stack, ori-
ginString));
                        case 'P' -> processRule(P(string, stack, ori-
ginString));
                        default -> {
                            if (m == x) {
                                stack = pop(stack);
                                string = next(string);
                            } else {
                                throw new InvalidStringError();
                            }
                        }
                    }
                }
            }
        } catch (InvalidStringError e) {
            System.out.println("Цепочка недопустима");
            return false;
        }
        System.out.println("Цепочка допустима");
        return true;
    }

    private void processRule(RuleResult result) {
        stack = result.stack();
        string = result.string();
    }

    private record RuleResult(String stack, String string) {}

    private static String hold(String string) {
        return string;
    }
```

```
    private static String next(String string) {
        return string.substring(1);
    }

    private String replace(String stack, String value) {
        return new StringBuilder(value).reverse().toString() + stack.sub-
string(1);
    }

    private static String pop(String stack) {
        return stack.equals("Δ") ? stack : stack.substring(1);
    }

    private void printStep(String string, String stack, String originString,
String rule) {
        System.out.println("-".repeat(30));
        System.out.printf("Текущая цепочка: %s%s%n",
                originString.substring(0, originString.length() -
string.length() + 1),
                stack.substring(0, stack.length() - 1));
        System.out.printf("Правило: %s%n", rule);
    }

    private RuleResult S(String string, String stack, String originString) {
        if (string.charAt(0) == 'a') {
            printStep(string, stack, originString, "1. S -> aFG");
            return new RuleResult(replace(stack, "GF"), next(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult G(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == 'a') {
            printStep(string, stack, originString, "2. G -> aFG");
            return new RuleResult(replace(stack, "GF"), next(string));
        } else if (firstChar == '┤' || firstChar == ']') {
            printStep(string, stack, originString, "3. G -> ε");
            return new RuleResult(pop(stack), hold(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult F(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '[') {
            printStep(string, stack, originString, "4. F -> [S]Q");
            return new RuleResult(replace(stack, "Q]S"), next(string));
```

```
        } else if (firstChar == '=') {
            printStep(string, stack, originString, "5. F -> =E");
            return new RuleResult(replace(stack, "E"), next(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult Q(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '[') {
            printStep(string, stack, originString, "6. Q -> [S]");
            return new RuleResult(replace(stack, "]S"), next(string));
        } else if (firstChar == 'a' || firstChar == ']' || firstChar == '┤') {
            printStep(string, stack, originString, "7. Q -> ε");
            return new RuleResult(pop(stack), hold(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult E(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '(') {
            printStep(string, stack, originString, "8. E -> (E)LV");
            return new RuleResult(replace(stack, "VL)E"), next(string));
        } else if (firstChar == 'a') {
            printStep(string, stack, originString, "8. E -> aLV");
            return new RuleResult(replace(stack, "VL"), next(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult V(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '+') {
            printStep(string, stack, originString, "10. V -> +TV");
            return new RuleResult(replace(stack, "VT"), next(string));
        } else if (firstChar == ')' || firstChar == 'a' || firstChar == '┤' ||
firstChar == ']') {
            printStep(string, stack, originString, "11. V -> ε");
            return new RuleResult(pop(stack), hold(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult T(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '(') {
            printStep(string, stack, originString, "12. T -> (E)L");
```

```
            return new RuleResult(replace(stack, "L)E"), next(string));
        } else if (firstChar == 'a') {
            printStep(string, stack, originString, "13. T -> aL");
            return new RuleResult(replace(stack, "L"), next(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult L(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '*') {
            printStep(string, stack, originString, "14. L -> *PL");
            return new RuleResult(replace(stack, "LP"), next(string));
        } else if (firstChar == '+' || firstChar == ')' || firstChar == 'a' ||
firstChar == '┤' || firstChar == ']') {
            printStep(string, stack, originString, "15. L -> ε");
            return new RuleResult(pop(stack), hold(string));
        }
        throw new InvalidStringError();
    }

    private RuleResult P(String string, String stack, String originString) {
        char firstChar = string.charAt(0);
        if (firstChar == '(') {
            printStep(string, stack, originString, "16. P -> (E)");
            return new RuleResult(replace(stack, ")E"), next(string));
        } else if (firstChar == 'a') {
            printStep(string, stack, originString, "17. P -> a");
            return new RuleResult(pop(stack), next(string));
        }
        throw new InvalidStringError();
    }

    public static void main(String[] args) {
        PushdownParser parser1 = new PushdownParser("a[a=a*a]a=(a+a)");
        parser1.parse();
        System.out.println();
        PushdownParser parser2 = new PushdownParser("a[a+a]a");
        parser2.parse();
        System.out.println();
        PushdownParser parser3 = new PushdownParser("bscsds");
        parser3.parse();
    }
}
```

Задание 10. Обработать цепочки из набора тестовых данных (см. п.6) программой-распознавателем.

Результат работы программы:

```
Путь применяемых правил для a[a=a*a]a=(a+a):
------------------------------
Текущая цепочка: S
Правило: 1. S -> aFG
------------------------------
Текущая цепочка: aFG
Правило: 4. F -> [S]Q
------------------------------
Текущая цепочка: a[S]QG
Правило: 1. S -> aFG
------------------------------
Текущая цепочка: a[aFG]QG
Правило: 5. F -> =E
------------------------------
Текущая цепочка: a[a=EG]QG
Правило: 8. E -> aLV
------------------------------
Текущая цепочка: a[a=aLVG]QG
Правило: 14. L -> *PL
------------------------------
Текущая цепочка: a[a=a*PLVG]QG
Правило: 17. P -> a
------------------------------
Текущая цепочка: a[a=a*aLVG]QG
Правило: 15. L -> ε
------------------------------
Текущая цепочка: a[a=a*aVG]QG
Правило: 11. V -> ε
------------------------------
Текущая цепочка: a[a=a*aG]QG
Правило: 3. G -> ε
------------------------------
Текущая цепочка: a[a=a*a]QG
Правило: 7. Q -> ε
------------------------------
Текущая цепочка: a[a=a*a]G
Правило: 2. G -> aFG
```

```
------------------------------
Текущая цепочка: a[a=a*a]aFG
Правило: 5. F -> =E
------------------------------
Текущая цепочка: a[a=a*a]a=EG
Правило: 8. E -> (E)LV
------------------------------
Текущая цепочка: a[a=a*a]a=(E)LVG
Правило: 8. E -> aLV
------------------------------
Текущая цепочка: a[a=a*a]a=(aLV)LVG
Правило: 15. L -> ε
------------------------------
Текущая цепочка: a[a=a*a]a=(aV)LVG
Правило: 10. V -> +TV
------------------------------
Текущая цепочка: a[a=a*a]a=(a+TV)LVG
Правило: 13. T -> aL
------------------------------
Текущая цепочка: a[a=a*a]a=(a+aLV)LVG
Правило: 15. L -> ε
------------------------------
Текущая цепочка: a[a=a*a]a=(a+aV)LVG
Правило: 11. V -> ε
------------------------------
Текущая цепочка: a[a=a*a]a=(a+a)LVG
Правило: 15. L -> ε
------------------------------
Текущая цепочка: a[a=a*a]a=(a+a)VG
Правило: 11. V -> ε
------------------------------
Текущая цепочка: a[a=a*a]a=(a+a)G
Правило: 3. G -> ε
Цепочка допустима
```

```
Путь применяемых правил для a[a+a]a:
------------------------------
Текущая цепочка: S
Правило: 1. S -> aFG
------------------------------
Текущая цепочка: aFG
Правило: 4. F -> [S]Q
------------------------------
Текущая цепочка: a[S]QG
Правило: 1. S -> aFG
Цепочка недопустима

Путь применяемых правил для bscsds:
Цепочка недопустима
```

Вывод: на этой лабораторной работе мы изучили и научились применять нисходящие методы обработки формальных языков.