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)

ИИТУС
БГТУ им. В.Г. Шухова

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4

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

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич

Белгород 2024 г.

Лабораторная работа №3

Регулярные языки и конечные распознаватели

Вариант 8

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

1. Преобразовать исходную КС-грамматику в LL(1)-грамматику.
   1. S→O;S
   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→-(E)
   12. P→a

   Устраним левую рекурсию в грамматике:

   Удалим самолеворекурсивное правило E→E+T
   S→O;S
   S→O;
   O→a[S]
   O→a[S][S]
   O→a=E
   E→TL_1
   L_1→+TL_1
   L_1→ε
   T→T*P
   T→P
   P→(E)
   P→-(E)
   P→a

   Удалим самолеворекурсивное правило T→T*P
   S→O;S
   S→O;
   O→a[S]
   O→a[S][S]
   O→a=E
   E→TL_1

T→PL_2
P→(E)
P→-(E)
P→a
L_1→+TL_1
L_1→ε
L_2→*PL_2
L_2→ε

В грамматике есть правила с общими префиксами, выполним левую факторизацию.
S→O;S_1
O→aS_2
E→TL_1
T→PL_2
P→(E)
P→-(E)
P→a
L_1→+TL_1
L_1→ε
L_2→*PL_2
L_2→ε
S_1→S
S_1→ε
S_2→[S]S_3
S_2→=E
S_3→[S]
S_3→ε

В грамматике нет правил с одинаковой левой частью, из правых частей которых выводятся цепочки, имеющие общий префикс.

Определим ВЫБОР для правил кс-грамматики:

Правило	ВЫБОР
S→O;S_1	a
O→aS_2	a
E→TL_1	(- a
T→PL_2	(- a
P→(E)	(
P→-(E)	-
P→a	a
L_1→+TL_1	+
L_1→ε	) ;
L_2→*PL_2	*
L_2→ε	+ ;)
S_1→S	a
S_1→ε	┤]
S_2→[S]S_3	[
S_2→=E	=
S_3→[S]	[
S_3→ε	;

Символ	ПЕРВ	СЛЕД
S	a	┤]
O	a	;
E	(- a	) ;
T	(- a	+ ;)
P	(- a	* + ;)
L_1	+ ε	) ;
L_2	* ε	+ ;)
S_1	a ε	┤]
S_2	[=	;
S_3	[ε	;

Так как пары правил с одинаковой левой частью и непустым пересечений множеств ВЫБОР нет, можно сказать, что грамматика преобразована к LL(1)-грамматике. Искомая грамматика:

1. S→O;S_1
2. O→aS_2
3. E→TL_1
4. T→PL_2
5. P→(E)
6. P→-(E)
7. P→a
8. L_1→+TL_1
9. L_1→ε
10. L_2→*PL_2
11. L_2→ε
12. S_1→S

13. S_1→ε
14. S_2→[S]S_3
15. S_2→=E
16. S_3→[S]
17. S_3→ε

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

Символ	ПЕРВ
S	a
O	a
E	(- a
T	(- a
P	(- a
L_1	+ ε
L_2	* ε
S_1	a ε
S_2	[=
S_3	[ε

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

Символ	СЛЕД
S	⊣]
O	;
E	) ;
T	+ ;)
P	* + ;)
L_1	) ;
L_2	+ ;)
S_1	⊣]
S_2	;
S_3	;

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

Правило	ВЫБОР
S→O;S_1	a
O→aS_2	a
E→TL_1	(- a
T→PL_2	(- a
P→(E)	(
P→-(E)	-
P→a	a
L_1→+TL_1	+
L_1→ε	) ;
L_2→*PL_2	*
L_2→ε	+ ;)
S_1→S	a
S_1→ε	┤]
S_2→[S]S_3	[
S_2→=E	=
S_3→[S]	[
S_3→ε	;

5. Написать программу-распознаватель методом рекурсивного спуска. Программа должна выводить последовательность номеров правил, применяемых при левом выводе обрабатываемой цепочки.
   Переобозначим символы в грамматике:
   1. S→O;C {a}
   2. O→aD {a}
   3. E→TA {(, -, a}
   4. T→PB {(, -, a}
   5. P→(E) {(}
   6. P→-(E) {-}
   7. P→a {a}
   8. A→+TA {+}
   9. A→ε {), ;}
   10. B→*PB {*}
   11. B→ε {+, ;,)}
   12. C→S {a}
   13. C→ε {┤,]}
   14. D→[S]F {[}
   15. D→=E {=}
   16. F→[S] {[}
   17. F→ε {;}

```
InvalidStringError = ValueError("Invalid string detected")



def process_terminal(data, terminal):
```

```
    if data[0] == terminal:
        data = data[1:]
    else:
        raise InvalidStringError

    return data


def S(data: str) -> str:
    if data[0] in ["a"]:
        print("    1. S -> O;C")
        data = O(data)
        data = process_terminal(data, ";")
        data = C(data)
    else:
        raise InvalidStringError

    return data


def O(data: str) -> str:
    if data[0] in ["a"]:
        print("    2. O -> aD")
        data = process_terminal(data, "a")
        data = D(data)
    else:
        raise InvalidStringError

    return data


def E(data: str) -> str:
    if data[0] in ["(", "-", "a"]:
        print("    3. E -> TA")
        data = T(data)
        data = A(data)
    else:
        raise InvalidStringError

    return data


def T(data: str) -> str:
    if data[0] in ["(", "-", "a"]:
        print("    4. T -> PB")
        data = P(data)
        data = B(data)
    else:
        raise InvalidStringError

    return data


def P(data: str) -> str:
```

```
    if data[0] in ["("]:
        print("    5. P -> (E)")
        data = process_terminal(data, "(")
        data = E(data)
        data = process_terminal(data, ")")
    elif data[0] in ["-"]:
        print("    6. P -> -(E)")
        data = process_terminal(data, "-")
        data = process_terminal(data, "(")
        data = E(data)
        data = process_terminal(data, ")")
    elif data[0] in ["a"]:
        print("    7. P -> a")
        data = process_terminal(data, "a")
    else:
        raise InvalidStringError

    return data


def A(data: str) -> str:
    if data[0] in ["+"]:
        print("    8. A -> +TA")
        data = process_terminal(data, "+")
        data = T(data)
        data = A(data)
    elif data[0] in [")", ";"]:
        print("    9. A -> ε")
        pass
    else:
        raise InvalidStringError

    return data


def B(data: str) -> str:
    if data[0] in ["*"]:
        print("    10. B -> *PB")
        data = process_terminal(data, "*")
        data = P(data)
        data = B(data)
    elif data[0] in ["+", ";", ")"]:
        print("    11. B -> ε")
        pass
    else:
        raise InvalidStringError

    return data


def C(data: str) -> str:
    if data[0] in ["a"]:
        print("    12. C -> S")
        data = S(data)
```

```
    elif data[0] in ["┤", "]"]:
        print("    13. C -> ε")
        pass
    else:
        raise InvalidStringError

    return data


def D(data: str) -> str:
    if data[0] in ["["]:
        print("    14. D -> [S]F")
        data = process_terminal(data, "[")
        data = S(data)
        data = process_terminal(data, "]")
        data = F(data)
    elif data[0] in ["="]:
        print("    15. D -> =E")
        data = process_terminal(data, "=")
        data = E(data)
    else:
        raise InvalidStringError

    return data


def F(data: str) -> str:
    if data[0] in ["["]:
        print("    16. F -> [S]")
        data = process_terminal(data, "[")
        data = S(data)
        data = process_terminal(data, "]")
    elif data[0] in [";"]:
        print("    17. F -> ε")
        pass
    else:
        raise InvalidStringError

    return data


def LL1(data):
    print(f"Трейсбек применяемых правил для {data}:")
    data += "┤"
    try:
        result = S(data)
    except ValueError:
        print("Цепочка невалидна")
        raise InvalidStringError

    if result == "┤":
        print("Цепочка валидна")
        return True
    else:
```

```
        print("Цепочка невалидна")
        raise InvalidStringError
```

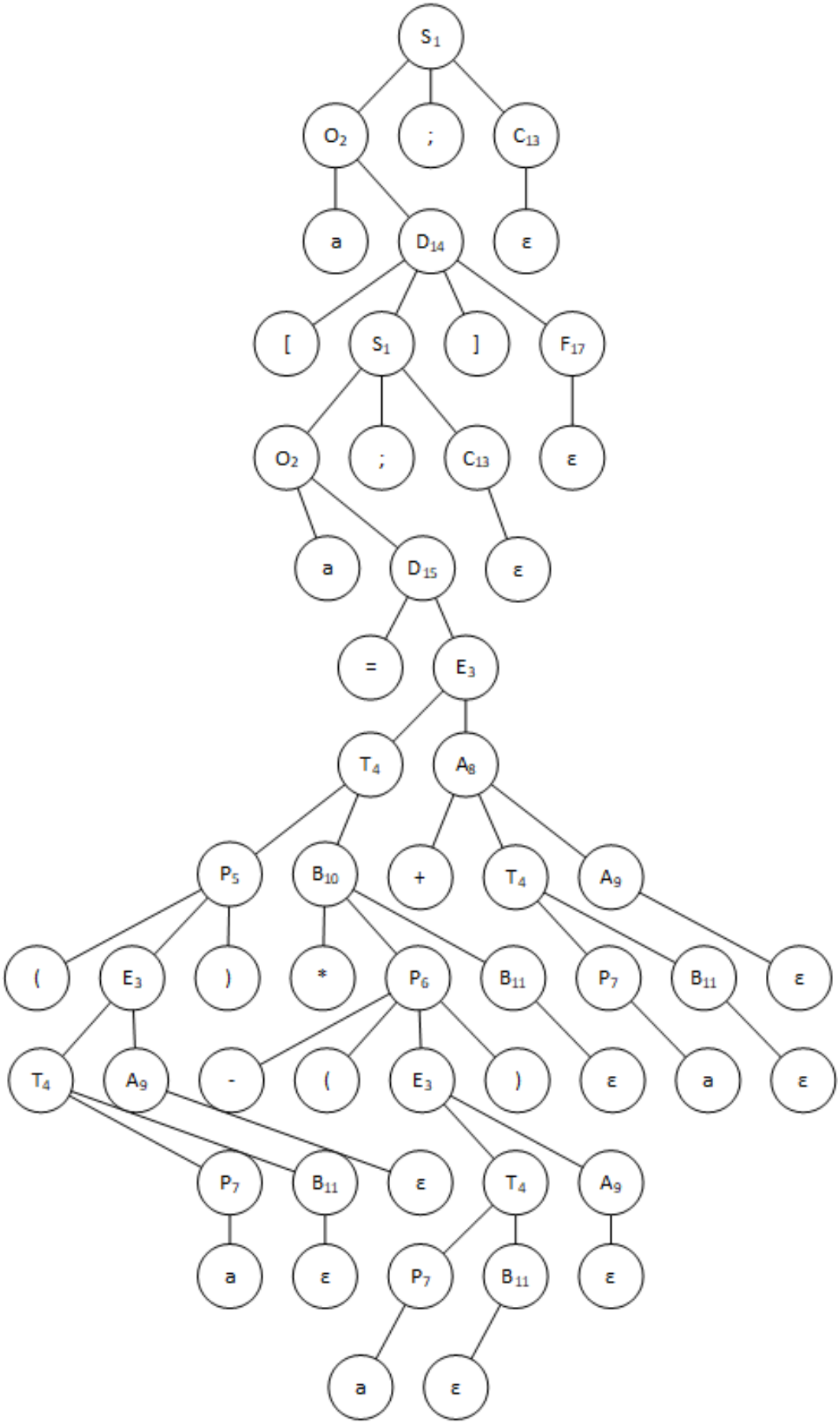
6. Сформировать наборы тестовых данных. Тестовые данные должны содержать цепочки, принадлежащие языку, заданному грамматикой, (допустимые цепочки) и цепочки, не принадлежащие языку. Для каждой допустимой цепочки построить дерево вывода и левый вывод. Каждое правило грамматики должно использоваться в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

   Тестовые данные:

   - **a[a=(a)*-(a)+a+-(a);];**
     Валидная цепочка

     Левый вывод:
     $S_1 \Rightarrow O_2;C \Rightarrow aD_{14};C \Rightarrow a[S_1]F;C \Rightarrow a[O_2;C]F;C \Rightarrow a[aD_{15};C]F;C \Rightarrow a[a = E_3;C]F;C \Rightarrow a[a = T_4A;C]F;C \Rightarrow a[a = P_5BA;C]F;C \Rightarrow a[a = (E_3)BA;C]F;C \Rightarrow a[a = (T_4A)BA;C]F;C \Rightarrow a[a = (P_7BA)BA;C]F;C \Rightarrow a[a = (aB_{11}A)BA;C]F;C \Rightarrow a[a = (aA_9)BA;C]F;C \Rightarrow a[a = (a)B_{10}A;C]F;C \Rightarrow a[a = (a) * P_6BA;C]F;C \Rightarrow a[a = (a) * -(E_3)BA;C]F;C \Rightarrow a[a = (a) * -(T_4A)BA;C]F;C \Rightarrow a[a = (a) * -(P_7BA)BA;C]F;C \Rightarrow a[a = (a) * -(aB_{11}A)BA;C]F;C \Rightarrow a[a = (a) * -(aA_9)BA;C]F;C \Rightarrow a[a = (a) * -(a)B_{11}A;C]F;C \Rightarrow a[a = (a) * -(a)A_8;C]F;C \Rightarrow a[a = (a) * -(a) + T_4A;C]F;C \Rightarrow a[a = (a) * -(a) + P_7BA;C]F;C \Rightarrow a[a = (a) * -(a) + aB_{11}A;C]F;C \Rightarrow a[a = (a) * -(a) + aA_9;C]F;C \Rightarrow a[a = (a) * -(a) + a;C_{13}]F;C \Rightarrow a[a = (a) * -(a) + a;]F_{17};C \Rightarrow a[a = (a) * -(a) + a;];C_{13} \Rightarrow a[a = (a) * -(a) + a;];$

     Дерево вывода:

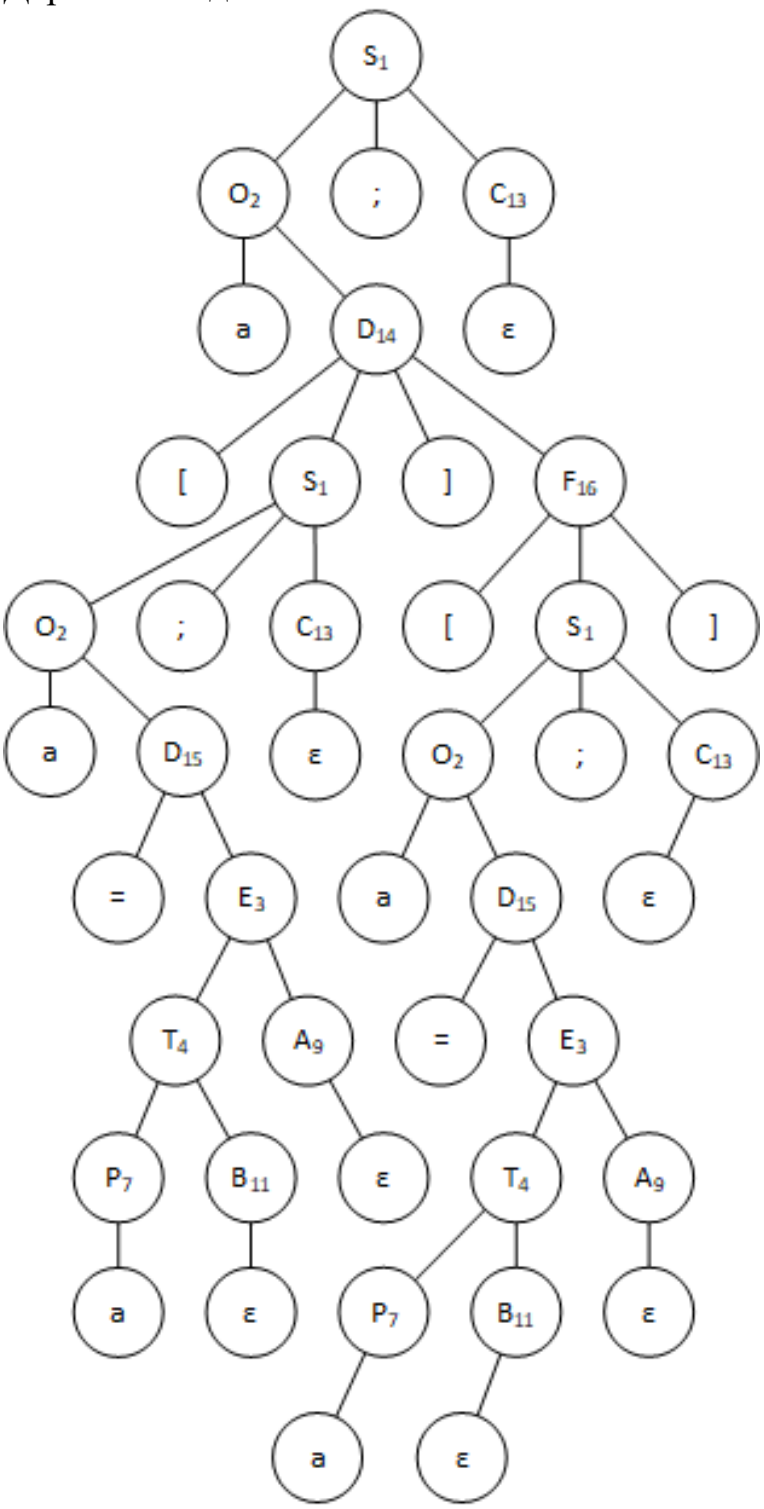


- **a[a=a;][a=a;];**
  Валидная цепочка

  Левый вывод:
  $S_1 \Rightarrow O_2;C \Rightarrow aD_{14};C \Rightarrow a[S_1]F;C \Rightarrow a[O_2;C]F;C \Rightarrow a[aD_{15};C]F;C \Rightarrow a[a = E_3;C]F;C \Rightarrow a[a = T_4A;C]F;C \Rightarrow a[a = P_7BA;C]F;C \Rightarrow a[a =$

$aB_{11}A;C]F;C \Rightarrow a[a = aA_9;C]F;C \Rightarrow a[a = a;C_{13}]F;C \Rightarrow a[a = a;]F_{16};C \Rightarrow a[a = a;][S_1];C \Rightarrow a[a = a;][O_2;C];C \Rightarrow a[a = a;][aD;C];C \Rightarrow a[a = a;][aD_{15};C];C \Rightarrow a[a = a;][a = E_3;C];C \Rightarrow a[a = a;][a = T_4A;C];C \Rightarrow a[a = a;][a = P_7BA;C];C \Rightarrow a[a = a;][a = aB_{11}A;C];C \Rightarrow a[a = a;][a = aA_9;C];C \Rightarrow a[a = a;][a = a;C_{13}];C \Rightarrow a[a = a;][a = a;];C_{13} \Rightarrow a[a = a;][a = a;];$

Дерево вывода:

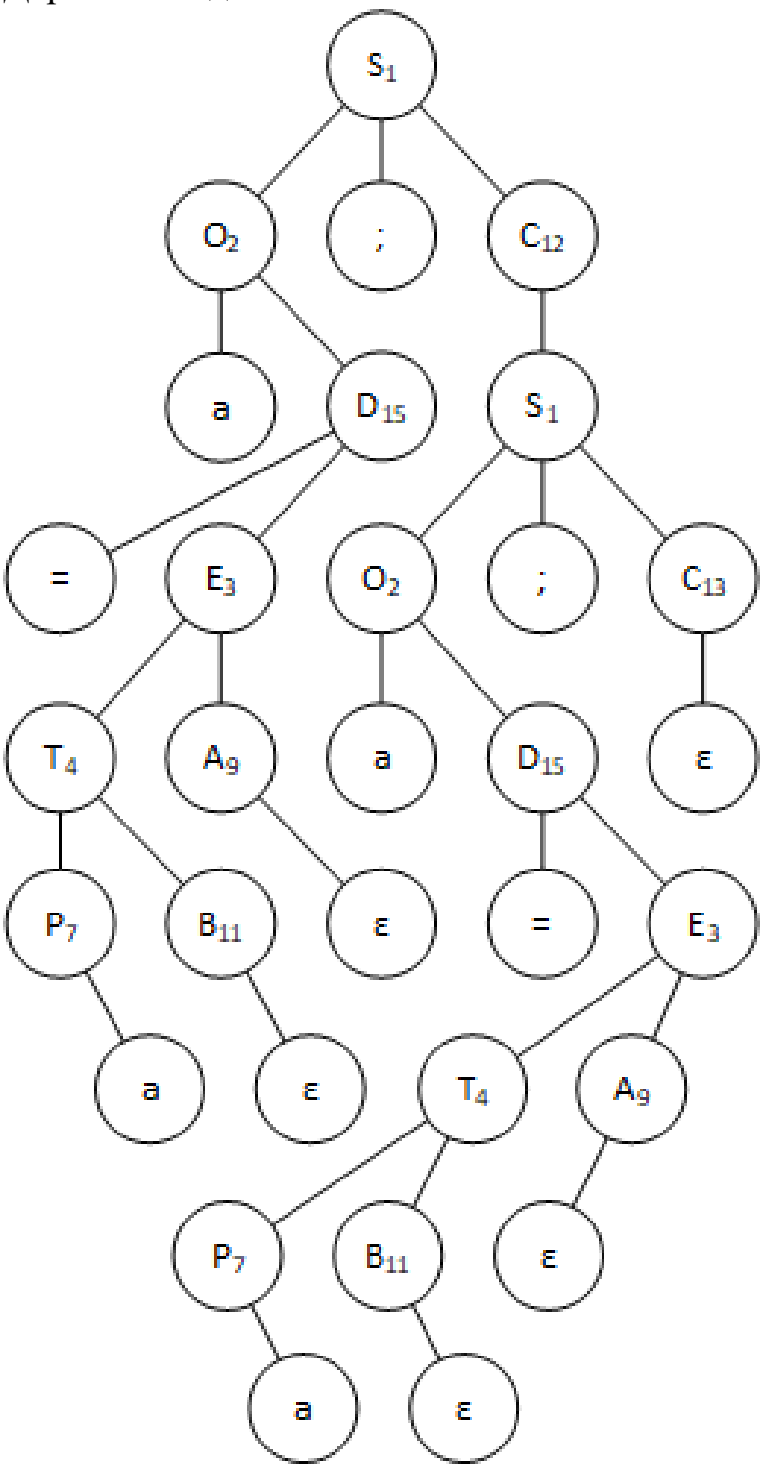


- **a=a;a=a;**
  Валидная цепочка

  Левый вывод:
  $S_1 \Rightarrow O_2;C \Rightarrow aD_{15};C \Rightarrow a = E_3;C \Rightarrow a = T_4A;C \Rightarrow a = P_7BA;C \Rightarrow a =$

$aB_{11}A; C \Rightarrow a = aA_9; C \Rightarrow a = a; C_{12} \Rightarrow a = a; S_1 \Rightarrow a = a; O_2; C \Rightarrow a = a; aD_{15}; C \Rightarrow a = a; a = E; C \Rightarrow a = a; a = E_3; C \Rightarrow a = a; a = T_4A; C \Rightarrow a = a; a = P_7BA; C \Rightarrow a = a; a = aB_{11}A; C \Rightarrow a = a; a = aA_9; C \Rightarrow a = a; a = a; C_{13} \Rightarrow a = a; a = a;$

Дерево вывода:



- **a[a=(a)*-(a)+a+-(a);]**
  Невалидная цепочка
- **a=[];**
  Невалидная цепочка
- **a=[a=a;][a=a;][a=a;][a=a;];**
  Невалидная цепочка

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

```
if __name__ == "__main__":
    assert LL1("a[a=(a)*-(a)+a;];")
    assert LL1("a[a=a;][a=a;];")
    assert LL1("a=a;a=a;")
    try:
        LL1("a[a=(a)*-(a)+a+-(a);]")
        assert False
    except ValueError:
        assert True
    try:
        LL1("a=[];")
        assert False
    except ValueError:
        assert True
    try:
        LL1("a=[a=a;][a=a;][a=a;][a=a;];")
        assert False
    except ValueError:
        assert True
```

Результат выполнения программы:

```
Трейсбек применяемых правил для a[a=(a)*-(a)+a;];:
    1. S -> O;C
    2. O -> aD
    14. D -> [S]F
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    5. P -> (E)
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    10. B -> *PB
    6. P -> -(E)
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    11. B -> ε
    8. A -> +TA
    4. T -> PB
    7. P -> a
    11. B -> ε
```

```
    9. A -> ε
    13. C -> ε
    17. F -> ε
    13. C -> ε
Цепочка валидна
Трейсбек применяемых правил для a[a=a;][a=a;];:
    1. S -> O;C
    2. O -> aD
    14. D -> [S]F
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    13. C -> ε
    16. F -> [S]
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    13. C -> ε
    13. C -> ε
Цепочка валидна
Трейсбек применяемых правил для a=a;a=a;:
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    12. C -> S
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    13. C -> ε
Цепочка валидна
Трейсбек применяемых правил для a[a=(a)*-(a)+a+-(a);]:
    1. S -> O;C
    2. O -> aD
    14. D -> [S]F
```

```
    1. S -> O;C
    2. O -> aD
    15. D -> =E
    3. E -> TA
    4. T -> PB
    5. P -> (E)
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    10. B -> *PB
    6. P -> -(E)
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    11. B -> ε
    8. A -> +TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    8. A -> +TA
    4. T -> PB
    6. P -> -(E)
    3. E -> TA
    4. T -> PB
    7. P -> a
    11. B -> ε
    9. A -> ε
    11. B -> ε
    9. A -> ε
    13. C -> ε
Цепочка невалидна
Трейсбек применяемых правил для a=[];:
    1. S -> O;C
    2. O -> aD
    15. D -> =E
Цепочка невалидна
Трейсбек применяемых правил для a=[a=a;][a=a;][a=a;][a=a;];:
    1. S -> O;C
    2. O -> aD
    15. D -> =E
Цепочка невалидна
```

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

	;	a	(	)	-	+	[	]	=	*	⊣
S		#1									
O		#2									
E		#3	#3		#3						
T		#4	#4		#4						
P		#7	#5		#6						
A	#9			#9		#8					
B	#9			#9		#9				#10	
C		#11						#9			#9
D							#12		#13		
F	#9						#14				
)				#7							
]								#7			
△											ДОПУСТ.

#1: ЗАМЕНИТЬ(C;O), ДЕРЖАТЬ
#2: ЗАМЕНИТЬ(D), СДВИГ
#3: ЗАМЕНИТЬ(AT), ДЕРЖАТЬ
#4: ЗАМЕНИТЬ(BP), ДЕРЖАТЬ
#5: ЗАМЕНИТЬ()E), СДВИГ
#6: ЗАМЕНИТЬ()E(), СДВИГ
#7: ВЫТОЛКНУТЬ, СДВИГ
#8: ЗАМЕНИТЬ(AT), СДВИГ
#9: ВЫТОЛКНУТЬ, ДЕРЖАТЬ
#10: ЗАМЕНИТЬ(BP), СДВИГ
#11: ЗАМЕНИТЬ(S), ДЕРЖАТЬ
#12: ЗАМЕНИТЬ(F]S), СДВИГ
#13: ЗАМЕНИТЬ(E), СДВИГ
#14: ЗАМЕНИТЬ(]S), СДВИГ

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

```
InvalidStringError = ValueError("Invalid string detected")


def hold(data: str) -> str:
    return data


def next(data: str) -> str:
    return data[1:]


def replace(stack: str, value: str) -> str:
    return value[::-1] + stack[1:]
```

```python
def pop(stack: str) -> str:
    return stack[1:] if stack != "Δ" else stack


def print_step(data, stack, origin_data, rule):
    print(f"*********\n"
          f"Текущая цепочка: {origin_data[0:(len(origin_data) - len(data) + 1)]}{stack[:-1]}\n"
          f"Правило: {rule}\n\n")


def shift_reduce_parser(data):
    print(f"Трейсбек применяемых правил для {data}:")
    try:
        origin_data = data
        data += "┤"
        stack = "SΔ"
        should_iter = -1
        while should_iter == -1:
            m = stack[0]
            x = data[0]
            if m == "Δ" and data == "┤":
                if data == "┤":
                    should_iter = 1
                else:
                    raise InvalidStringError
            elif m == "S":
                stack, data = S(data, stack, origin_data)
            elif m == "O":
                stack, data = O(data, stack, origin_data)
            elif m == "E":
                stack, data = E(data, stack, origin_data)
            elif m == "T":
                stack, data = T(data, stack, origin_data)
            elif m == "P":
                stack, data = P(data, stack, origin_data)
            elif m == "A":
                stack, data = A(data, stack, origin_data)
            elif m == "B":
                stack, data = B(data, stack, origin_data)
            elif m == "C":
                stack, data = C(data, stack, origin_data)
            elif m == "D":
                stack, data = D(data, stack, origin_data)
            elif m == "F":
                stack, data = F(data, stack, origin_data)
            elif m == x:
                (stack, data) = (pop(stack), next(data))
            else:
                raise InvalidStringError
    except ValueError:
        print("Цепочка невалидна")
        raise InvalidStringError

    print("Цепочка валидна")
```

```
    return True


def S(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["a"]:
        print_step(data, stack, origin_data, "    1. S -> O;C")
        # 1
        return replace(stack, "C;O"), hold(data)

    raise InvalidStringError


def O(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["a"]:
        print_step(data, stack, origin_data, "    2. O -> aD")
        # 2
        return replace(stack, "D"), next(data)

    raise InvalidStringError


def E(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["(", "-", "a"]:
        print_step(data, stack, origin_data, "    3. E -> TA")
        # 3
        return replace(stack, "AT"), hold(data)

    raise InvalidStringError


def T(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["(", "-", "a"]:
        print_step(data, stack, origin_data, "    4. T -> PB")
        # 4
        return replace(stack, "BP"), hold(data)

    raise InvalidStringError


def P(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["("]:
        print_step(data, stack, origin_data, "    5. P -> (E)")
        # 5
        return replace(stack, ")E"), next(data)
    elif data[0] in ["-"]:
        print_step(data, stack, origin_data, "    6. P -> -(E)")
        # 6
        return replace(stack, ")E("), next(data)
    elif data[0] in ["a"]:
        print_step(data, stack, origin_data, "    7. P -> a")
        # 7
        return pop(stack), next(data)
```

```
    raise InvalidStringError


def A(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["+"]:
        print_step(data, stack, origin_data, "    8. A -> +TA")
        # 8
        return replace(stack, "AT"), next(data)
    elif data[0] in [")", ";"]:
        print_step(data, stack, origin_data, "    9. A -> ε")
        # 9
        return pop(stack), hold(data)

    raise InvalidStringError


def B(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["*"]:
        print_step(data, stack, origin_data, "    10. B -> *PB")
        # 10
        return replace(stack, "BP"), next(data)
    elif data[0] in ["+", ";", ")"]:
        print_step(data, stack, origin_data, "    11. B -> ε")
        # 9
        return pop(stack), hold(data)

    raise InvalidStringError


def C(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["a"]:
        print_step(data, stack, origin_data, "    12. C -> S")
        # 11
        return replace(stack, "S"), hold(data)
    elif data[0] in ["|", "]"]:
        print_step(data, stack, origin_data, "    13. C -> ε")
        # 9
        return pop(stack), hold(data)

    raise InvalidStringError


def D(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["["]:
        print_step(data, stack, origin_data, "    14. D -> [S]F")
        # 12
        return replace(stack, "F]S"), next(data)
    elif data[0] in ["="]:
        print_step(data, stack, origin_data, "    15. D -> =E")
        # 13
        return replace(stack, "E"), next(data)

    raise InvalidStringError
```

```
def F(data: str, stack: str, origin_data: str) -> (str, str):
    if data[0] in ["["]:
        print_step(data, stack, origin_data, "    16. F -> [S]")
        # 14
        return replace(stack, "]S"), next(data)
    elif data[0] in [";"]:
        print_step(data, stack, origin_data, "    17. F -> ε")
        # 9
        return pop(stack), hold(data)

    raise InvalidStringError


if __name__ == "__main__":
    assert shift_reduce_parser("a[a=(a)*-(a)+a;];")
    assert shift_reduce_parser("a[a=a;][a=a;];")
    assert shift_reduce_parser("a=a;a=a;")
    try:
        shift_reduce_parser("a[a=(a)*-(a)+a+-(a);]")
        assert False
    except ValueError:
        assert True
    try:
        shift_reduce_parser("a=[];")
        assert False
    except ValueError:
        assert True
    try:
        shift_reduce_parser("a=[a=a;][a=a;][a=a;][a=a;];")
        assert False
    except ValueError:
        assert True
```

Результат выполнения программы:

```
Трейсбек применяемых правил для a[a=(a)*-(a)+a;];:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     14. D -> [S]F
```

```
*********
Текущая цепочка: a[S]F;C
Правило:     1. S -> O;C


*********
Текущая цепочка: a[O;C]F;C
Правило:     2. O -> aD


*********
Текущая цепочка: a[aD;C]F;C
Правило:     15. D -> =E


*********
Текущая цепочка: a[a=E;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=TA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=PBA;C]F;C
Правило:     5. P -> (E)


*********
Текущая цепочка: a[a=(E)BA;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=(TA)BA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(PBA)BA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(aBA)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(aA)BA;C]F;C
Правило:     9. A -> ε
```

```
*********
Текущая цепочка: a[a=(a)BA;C]F;C
Правило:     10. B -> *PB


*********
Текущая цепочка: a[a=(a)*PBA;C]F;C
Правило:     6. P -> -(E)


*********
Текущая цепочка: a[a=(a)*-(E)BA;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=(a)*-(TA)BA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(a)*-(PBA)BA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(a)*-(aBA)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(aA)BA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)*-(a)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)A;C]F;C
Правило:     8. A -> +TA


*********
Текущая цепочка: a[a=(a)*-(a)+TA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(a)*-(a)+PBA;C]F;C
```

```
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(a)*-(a)+aBA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+aA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a;C]F;C
Правило:     13. C -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a;]F;C
Правило:     17. F -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a;];C
Правило:     13. C -> ε


Цепочка валидна
Трейсбек применяемых правил для a[a=a;][a=a;];:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     14. D -> [S]F


*********
Текущая цепочка: a[S]F;C
Правило:     1. S -> O;C


*********
Текущая цепочка: a[O;C]F;C
Правило:     2. O -> aD
```

```
*********
Текущая цепочка: a[aD;C]F;C
Правило:     15. D -> =E


*********
Текущая цепочка: a[a=E;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=TA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=PBA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=aBA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=aA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=a;C]F;C
Правило:     13. C -> ε


*********
Текущая цепочка: a[a=a;]F;C
Правило:     16. F -> [S]


*********
Текущая цепочка: a[a=a;][S];C
Правило:     1. S -> O;C


*********
Текущая цепочка: a[a=a;][O;C];C
Правило:     2. O -> aD


*********
Текущая цепочка: a[a=a;][aD;C];C
Правило:     15. D -> =E
```

```
*********
Текущая цепочка: a[a=a;][a=E;C];C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=a;][a=TA;C];C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=a;][a=PBA;C];C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=a;][a=aBA;C];C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=a;][a=aA;C];C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=a;][a=a;C];C
Правило:     13. C -> ε


*********
Текущая цепочка: a[a=a;][a=a;];C
Правило:     13. C -> ε


Цепочка валидна
Трейсбек применяемых правил для a=a;a=a;:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     15. D -> =E
```

```
*********
Текущая цепочка: a=E;C
Правило:     3. E -> TA


*********
Текущая цепочка: a=TA;C
Правило:     4. T -> PB


*********
Текущая цепочка: a=PBA;C
Правило:     7. P -> a


*********
Текущая цепочка: a=aBA;C
Правило:     11. B -> ε


*********
Текущая цепочка: a=aA;C
Правило:     9. A -> ε


*********
Текущая цепочка: a=a;C
Правило:     12. C -> S


*********
Текущая цепочка: a=a;S
Правило:     1. S -> O;C


*********
Текущая цепочка: a=a;O;C
Правило:     2. O -> aD


*********
Текущая цепочка: a=a;aD;C
Правило:     15. D -> =E


*********
Текущая цепочка: a=a;a=E;C
Правило:     3. E -> TA


*********
Текущая цепочка: a=a;a=TA;C
Правило:     4. T -> PB
```

```
*********
Текущая цепочка: a=a;a=PBA;C
Правило:     7. P -> a


*********
Текущая цепочка: a=a;a=aBA;C
Правило:     11. B -> ε


*********
Текущая цепочка: a=a;a=aA;C
Правило:     9. A -> ε


*********
Текущая цепочка: a=a;a=a;C
Правило:     13. C -> ε


Цепочка валидна
Трейсбек применяемых правил для a[a=(a)*-(a)+a+-(a);]:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     14. D -> [S]F


*********
Текущая цепочка: a[S]F;C
Правило:     1. S -> O;C


*********
Текущая цепочка: a[O;C]F;C
Правило:     2. O -> aD


*********
Текущая цепочка: a[aD;C]F;C
Правило:     15. D -> =E


*********
```

```
Текущая цепочка: a[a=E;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=TA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=PBA;C]F;C
Правило:     5. P -> (E)


*********
Текущая цепочка: a[a=(E)BA;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=(TA)BA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(PBA)BA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(aBA)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(aA)BA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)BA;C]F;C
Правило:     10. B -> *PB


*********
Текущая цепочка: a[a=(a)*PBA;C]F;C
Правило:     6. P -> -(E)


*********
Текущая цепочка: a[a=(a)*-(E)BA;C]F;C
Правило:     3. E -> TA
```

```
*********
Текущая цепочка: a[a=(a)*-(TA)BA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(a)*-(PBA)BA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(a)*-(aBA)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(aA)BA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)*-(a)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)A;C]F;C
Правило:     8. A -> +TA


*********
Текущая цепочка: a[a=(a)*-(a)+TA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(a)*-(a)+PBA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(a)*-(a)+aBA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+aA;C]F;C
Правило:     8. A -> +TA


*********
Текущая цепочка: a[a=(a)*-(a)+a+TA;C]F;C
Правило:     4. T -> PB
```

```
*********
Текущая цепочка: a[a=(a)*-(a)+a+PBA;C]F;C
Правило:     6. P -> -(E)


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(E)BA;C]F;C
Правило:     3. E -> TA


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(TA)BA;C]F;C
Правило:     4. T -> PB


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(PBA)BA;C]F;C
Правило:     7. P -> a


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(aBA)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(aA)BA;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(a)BA;C]F;C
Правило:     11. B -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(a)A;C]F;C
Правило:     9. A -> ε


*********
Текущая цепочка: a[a=(a)*-(a)+a+-(a);C]F;C
Правило:     13. C -> ε


Цепочка невалидна
Трейсбек применяемых правил для a=[];:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
```

```
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     15. D -> =E


Цепочка невалидна
Трейсбек применяемых правил для a=[a=a;][a=a;][a=a;][a=a;];:
*********
Текущая цепочка: S
Правило:     1. S -> O;C


*********
Текущая цепочка: O;C
Правило:     2. O -> aD


*********
Текущая цепочка: aD;C
Правило:     15. D -> =E


Цепочка невалидна
```

Вывод: в ходе лабораторной работы изучили и научились применять нисходящие методы обработки формальных языков.