

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
Дмитриев Андрей
Александрович

Проверил:
Рязанов Юрий Дмитриевич

Белгород 2024 г.

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

Вариант 2:

Вариант 2
1. $S \rightarrow S;O$
2. $S \rightarrow O$
3. $O \rightarrow Y[S]$
4. $O \rightarrow Y[S][S]$
5. $O \rightarrow a=E$
6. $Y \rightarrow a=a$
7. $Y \rightarrow a<a$
8. $Y \rightarrow !(Y)$
9. $E \rightarrow (E+E)$
10. $E \rightarrow (E^*E)$
11. $E \rightarrow a$

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

1. $S \rightarrow S;O$
2. $S \rightarrow O$
3. $O \rightarrow Y[S]$
4. $O \rightarrow Y[S][S]$
5. $O \rightarrow a=E$
6. $Y \rightarrow a=a$
7. $Y \rightarrow a<a$
8. $Y \rightarrow !(Y)$
9. $E \rightarrow (E+E)$
10. $E \rightarrow (E^*E)$
11. $E \rightarrow a$

Устраним самолеворекурсивное правило $S \rightarrow S;O$:

- $$\begin{aligned} S &\rightarrow OS' \\ S' &\rightarrow ;OS' \\ S' &\rightarrow \epsilon \\ O &\rightarrow Y[S] \\ O &\rightarrow Y[S][S] \\ O &\rightarrow a=E \\ Y &\rightarrow a=a \\ Y &\rightarrow a<a \\ Y &\rightarrow !(Y) \\ E &\rightarrow (E+E) \\ E &\rightarrow (E^*E) \\ E &\rightarrow a \end{aligned}$$

Устраним левую рекурсию:

$$\begin{aligned}
S &\rightarrow OS' \\
S' &\rightarrow ;OS' \\
S' &\rightarrow \epsilon \\
O &\rightarrow Y[S] \\
O &\rightarrow Y[S][S] \\
O &\rightarrow a=E \\
Y &\rightarrow a=a \\
Y &\rightarrow a<a \\
Y &\rightarrow !(Y) \\
E &\rightarrow (E+E) \\
E &\rightarrow (E*E) \\
E &\rightarrow a
\end{aligned}$$

Имеются правила с идентичными префиксами, факторизуем эти правила:

$$\begin{aligned}
S &\rightarrow OS' \\
S' &\rightarrow ;OS' \\
S' &\rightarrow \epsilon \\
O &\rightarrow Y[S]O' \\
O' &\rightarrow [S] \\
O' &\rightarrow \epsilon \\
O &\rightarrow a=E \\
Y &\rightarrow aY' \\
Y' &\rightarrow =a \\
Y' &\rightarrow <a \\
Y &\rightarrow !(Y) \\
E &\rightarrow (EE' \\
E' &\rightarrow +E) \\
E' &\rightarrow *E) \\
E &\rightarrow a
\end{aligned}$$

Пересечение множеств ВЫБОР($O \rightarrow Y[S]O'$) и ВЫБОР($O \rightarrow a=E$) даёт непустое множество, устраним путём введения правил: $O \rightarrow aO_1$, $O_1 \rightarrow Y'[S]O'$ и $O_1 \rightarrow =E$:

$$\begin{aligned}
S &\rightarrow OS' \\
S' &\rightarrow ;OS' \\
S' &\rightarrow \epsilon \\
O &\rightarrow aO_1 \\
O &\rightarrow !(Y)[S]O' \\
O_1 &\rightarrow Y'[S]O' \\
O_1 &\rightarrow =E \\
O' &\rightarrow [S] \\
O' &\rightarrow \epsilon \\
Y &\rightarrow aY' \\
Y' &\rightarrow =a \\
Y' &\rightarrow <a \\
Y &\rightarrow !(Y) \\
E &\rightarrow (EE' \\
E' &\rightarrow +E) \\
E' &\rightarrow *E) \\
E &\rightarrow a
\end{aligned}$$

Пересечение множеств $\text{ВЫБОР}(O1 \rightarrow Y'[S]O')$ и $\text{ВЫБОР}(O1 \rightarrow =E)$ даёт непустое множество, устраним путём введения правил: $O1 \rightarrow =O2$, $O2 \rightarrow E$ и $O2 \rightarrow a[S]O'$:

$S \rightarrow OS'$
 $S' \rightarrow ;OS'$
 $S' \rightarrow \epsilon$
 $O \rightarrow aO1$
 $O \rightarrow !(Y)[S]O'$
 $O1 \rightarrow =O2$
 $O1 \rightarrow <a[S]O'$
 $O2 \rightarrow E$
 $O2 \rightarrow a[S]O'$
 $O' \rightarrow [S]$
 $O' \rightarrow \epsilon$
 $Y \rightarrow aY'$
 $Y' \rightarrow =a$
 $Y' \rightarrow <a$
 $Y \rightarrow !(Y)$
 $E \rightarrow (EE'$
 $E' \rightarrow +E)$
 $E' \rightarrow *E)$
 $E \rightarrow a$

Пересечение множеств $\text{ВЫБОР}(O2 \rightarrow E)$ и $\text{ВЫБОР}(O2 \rightarrow a[S]O')$ даёт непустое множество, устраним путём введения правил: $O2 \rightarrow aO3$, $O3 \rightarrow [S]O'$ и $O3 \rightarrow \epsilon$ – и получим LL(1) грамматика:

$S \rightarrow OS'$
 $S' \rightarrow ;OS'$
 $S' \rightarrow \epsilon$
 $O \rightarrow aO1$
 $O \rightarrow !(Y)[S]O'$
 $O1 \rightarrow =O2$
 $O1 \rightarrow <a[S]O'$
 $O2 \rightarrow (EE'$
 $O2 \rightarrow aO3$
 $O3 \rightarrow [S]O'$
 $O3 \rightarrow \epsilon$
 $O' \rightarrow [S]$
 $O' \rightarrow \epsilon$
 $Y \rightarrow aY'$
 $Y' \rightarrow =a$
 $Y' \rightarrow <a$
 $Y \rightarrow !(Y)$
 $E \rightarrow (EE'$
 $E' \rightarrow +E)$
 $E' \rightarrow *E)$
 $E \rightarrow a$

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

Символ	ПЕРВ
S	a !
S`	; e
O	a !
O1	< =
O2	(a
O3	[e
O`	[e
Y	a !
Y`	= <
E	(a
E`	+ *

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

Символ	СЛЕД
S	{ }
S`	{ }
O	; { }
O1	; { }
O2	; { }
O3	; { }
O`	; { }
Y)
Y`)
E	+ *)
E`	+ *)

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

Правило	ВЫБОР
$S \rightarrow OS'$	a !
$S' \rightarrow ;OS'$;
$S' \rightarrow \epsilon$	{ }
$O \rightarrow aO1$	a
$O \rightarrow !(Y)[S]O'$!
$O1 \rightarrow =O2$	=
$O1 \rightarrow <a[S]O'$	<
$O2 \rightarrow (EE'$	(
$O2 \rightarrow aO3$	a
$O3 \rightarrow [S]O'$	[
$O3 \rightarrow \epsilon$; { }
$O' \rightarrow [S]$	[
$O' \rightarrow \epsilon$; { }
$Y \rightarrow aY'$	a
$Y' \rightarrow =a$	=
$Y' \rightarrow <a$	<
$Y \rightarrow !(Y)$!

$E \rightarrow (EE'$	$($
$E' \rightarrow +E)$	$+$
$E' \rightarrow *E)$	$*$
$E \rightarrow a$	a

Задание 5. Написать программу-распознаватель методом рекурсивного спуска.

Программа должна выводить последовательность номеров правил, применяемых при левом выводе обрабатываемой цепочки.

```
G_TERMINALS = ":[a=<!()+*]"

class ParseException(Exception):
    def __init__(self):
        print("incorrect")

class Recurse:

    def __init__(self, chain: str):
        self.err_flag = False
        self.chain = chain + '|'
        self.non_term = {
            'S': self.S, # S
            'Z': self.St, # S`
            'O': self.O, # O
            '1': self.O1, # O1
            '2': self.O2, # O2
            '3': self.O3, # O3
            '0': self.Ot, # O`
            'Y': self.Y, # Y
            '7': self.Yt, # Y`
            'E': self.E, # E
            '8': self.Et, # E`
        }

    def S(self):
        CHANGE = {
            'a': 'OZ',
            '!': 'OZ',
        }
        try:
            in_sym = self.chain[0]
            print(f'S => {CHANGE[in_sym]}')
            self.parse(CHANGE[in_sym])
        except Exception:
            self.err_flag = True

    def St(self):
        CHANGE = {
            ';': ';OZ',
        }
        try:
            in_sym = self.chain[0]
            print(f'S` => {CHANGE[in_sym]}')
            self.parse(CHANGE[in_sym])
        except Exception:
            if self.chain[0] in '|':
                print(f'S` => e')
            else:
                self.err_flag = True

    def O(self):
        CHANGE = {
            'a': 'a1',
            '!': '!(Y)[S]O'
        }
        try:
```

```

        in_sym = self.chain[0]
        print(f'O => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def O1(self):
    CHANGE = {
        '=': '=2',
        '<': '<a[S]0'
    }
    try:
        in_sym = self.chain[0]
        print(f'O1 => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def O2(self):
    CHANGE = {
        '(': '(E8',
        'a': 'a3'
    }
    try:
        in_sym = self.chain[0]
        print(f'O2 => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def O3(self):
    CHANGE = {
        '[': '[S]0'
    }
    try:
        in_sym = self.chain[0]
        print(f'O3 => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        if self.chain[0] in ';|':
            print(f'O3 => e')
        else:
            self.err_flag = True

def Ot(self):
    CHANGE = {
        '[': '[S]'
    }
    try:
        in_sym = self.chain[0]
        print(f'O` => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        if self.chain[0] in ';|':
            print(f'O` => e')
        else:
            self.err_flag = True

def Y(self):
    CHANGE = {
        'a': 'a7',
        '!': '!(Y)'
    }
    try:
        in_sym = self.chain[0]
        print(f'Y => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

```

```

def Yt(self):
    CHANGE = {
        '=': '=a',
        '<': '<a'
    }
    try:
        in_sym = self.chain[0]
        print(f'Y` => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def E(self):
    CHANGE = {
        '(': '(E8',
        'a': 'a'
    }
    try:
        in_sym = self.chain[0]
        print(f'E => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def Et(self):
    CHANGE = {
        '+': '+E)',
        '*': '*E)'
    }
    try:
        in_sym = self.chain[0]
        print(f'E` => {CHANGE[in_sym]}')
        self.parse(CHANGE[in_sym])
    except Exception:
        self.err_flag = True

def parse(self, u: str):
    if self.err_flag:
        return

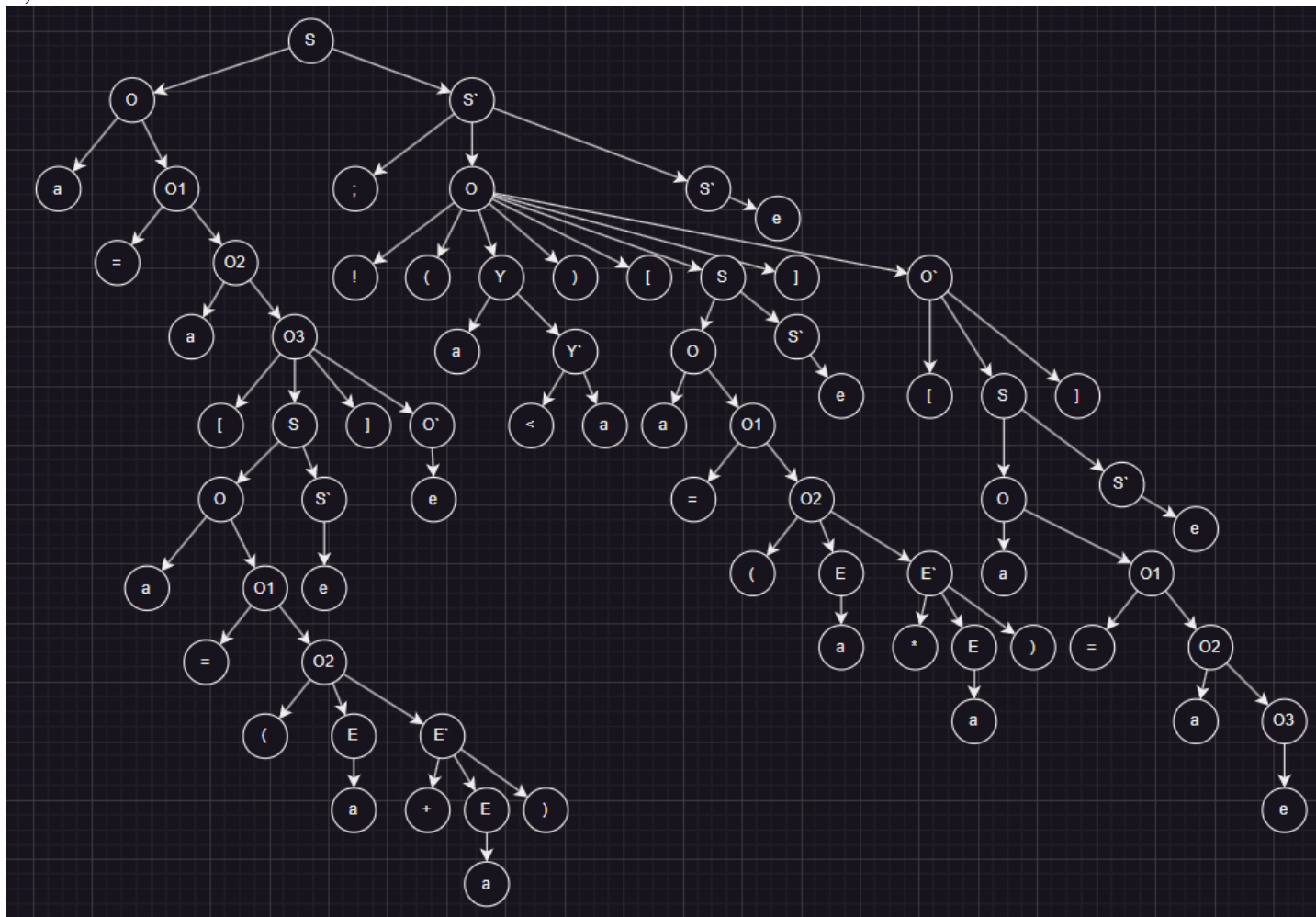
    v = u
    while v:
        X = v[0]
        z = v[1::]
        if X in G_TERMINALS:
            if X != self.chain[0]:
                raise Exception
            else:
                self.chain = self.chain[1::]
        else:
            self.non_term[X]()
        v = z

```

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.

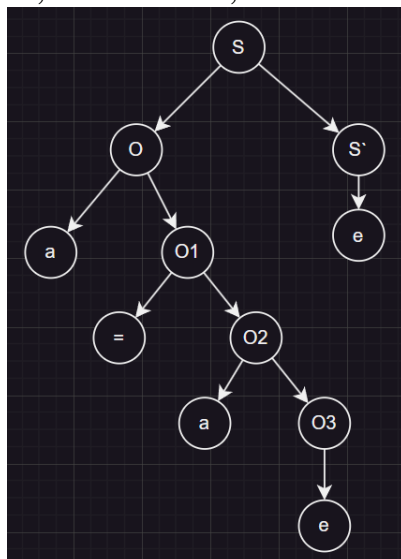
Допустимая цепочка: $a=a[a=(a+a)];!(a<a)[a=(a*a)][a=a]$

Левый вывод: $S \Rightarrow OS'; O \Rightarrow aO1; O1 \Rightarrow =O2; O2 \Rightarrow aO3; O3 \Rightarrow [S]O'; S \Rightarrow OS'; O \Rightarrow aO1; O1 \Rightarrow =O2; O2 \Rightarrow (EE'; E \Rightarrow a; E' \Rightarrow +E); E \Rightarrow a; S' \Rightarrow e; O' \Rightarrow e; S' \Rightarrow ;OS'; O \Rightarrow !(Y)[S]O'; Y \Rightarrow aY'; Y' \Rightarrow <a; S \Rightarrow OS'; O \Rightarrow aO1; O1 \Rightarrow =O2; O2 \Rightarrow (EE'; E \Rightarrow a; E' \Rightarrow *E); E \Rightarrow a; S' \Rightarrow e; O' \Rightarrow [S]; S \Rightarrow OS'; O \Rightarrow aO1; O1 \Rightarrow =O2; O2 \Rightarrow aO3; O3 \Rightarrow e; S' \Rightarrow e; S' \Rightarrow e$



Допустимая цепочка: $a=a$

Левый вывод: $S \Rightarrow OS'; O \Rightarrow aO1; O1 \Rightarrow =O2; O2 \Rightarrow aO3; O3 \Rightarrow e; S' \Rightarrow e$



Недопустимая цепочка: $a=a[a=(a+a)];$

Недопустимая цепочка: $a=a[$

Недопустимая цепочка: $!(a<a$

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

```
S -> OS`
O -> aO1
O1 -> =O2
O2 -> aO3
O3 -> [S]O`
S -> OS`
O -> aO1
O1 -> =O2
O2 -> (EE`
E -> a
E` -> +E)
E -> a
S` -> e
O` -> e
S` -> ;OS`
O -> !(Y)[S]O`
Y -> aY`
Y` -> <a
S -> OS`
O -> aO1
O1 -> =O2
O2 -> (EE`
E -> a
E` -> *E)
E -> a
S` -> e
O` -> [S]
S -> OS`
O -> aO1
O1 -> =O2
O2 -> aO3
O3 -> e
S` -> e
S` -> e
correct

S -> OS`
O -> aO1
O1 -> =O2
O2 -> aO3
O3 -> e
S` -> e
correct

S -> OS`
O -> aO1
O1 -> =O2
O2 -> aO3
O3 -> [S]O`
O3 -> e
S` -> e
incorrect

S -> OS`
O -> !(Y)[S]O`
Y -> aY`
Y` -> <a
S` -> e
incorrect
```

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

Имеем LL(1) грамматику:

1. $S \rightarrow OS'$ $a !$
2. $S' \rightarrow ;OS'$ $;$
3. $S' \rightarrow \epsilon$ $\downarrow]$
4. $O \rightarrow aO1$ a
5. $O \rightarrow !(Y)[S]O'$ $!$
6. $O1 \rightarrow =O2$ $=$
7. $O1 \rightarrow <a[S]O'$ $<$
8. $O2 \rightarrow (EE'$ $($
9. $O2 \rightarrow aO3$ a
10. $O3 \rightarrow [S]O'$ $[$
11. $O3 \rightarrow \epsilon$ $;\downarrow]$
12. $O' \rightarrow [S]$ $[$
13. $O' \rightarrow \epsilon$ $;\downarrow]$
14. $Y \rightarrow aY'$ a
15. $Y' \rightarrow =a$ $=$
16. $Y' \rightarrow <a$ $<$
17. $Y \rightarrow !(Y)$ $!$
18. $E \rightarrow (EE'$ $($
19. $E' \rightarrow +E$ $+$
20. $E' \rightarrow *E$ $*$
21. $E \rightarrow a$ a

МП-распознаватель:

	;	[]	a	=	<	!	()	+	*	\downarrow
S				#1			#1					
S'	#2		#3									#3
O				#4			#5					
O1					#6	#7						
O2				#9				#8				
O3	#11	#10	#11									#11
O'	#13	#12	#13									#13
Y				#14			#17					
Y'					#15	#16						
E				#21				#18				
E'										#19	#20	
[ВЫГ СД										
]			ВЫГ СД									
a				ВЫГ СД								
(ВЫГ СД				
)									ВЫГ СД			
Δ												ДОП

#1 ЗАМЕНИТЬ(S'O), держать

#2 ЗАМЕНИТЬ(S'O), сдвиг

#3 вытолкнуть, держать

#4 ЗАМЕНИТЬ(O1), сдвиг

#5 ЗАМЕНИТЬ(O`]S[]Y(), сдвиг
 #6 ЗАМЕНИТЬ(O2), сдвиг
 #7 ЗАМЕНИТЬ(O`]S[a), сдвиг
 #8 ЗАМЕНИТЬ(E`E), сдвиг
 #9 ЗАМЕНИТЬ(O3), сдвиг
 #10 ЗАМЕНИТЬ(O`]S), сдвиг
 #11 вытолкнуть, держать
 #12 ЗАМЕНИТЬ(]S), сдвиг
 #13 вытолкнуть, держать
 #14 ЗАМЕНИТЬ(Y`), сдвиг
 #15 ЗАМЕНИТЬ(а), сдвиг
 #16 ЗАМЕНИТЬ(а), сдвиг
 #17 ЗАМЕНИТЬ()Y(), сдвиг
 #18 ЗАМЕНИТЬ(E`E), сдвиг
 #19 ЗАМЕНИТЬ()E), сдвиг
 #20 ЗАМЕНИТЬ()E), сдвиг
 #21 ЗАМЕНИТЬ(), 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.

```
G_TERMINALS = ";[]a=<!(*)+*"
```

```

RULES = [
    ('S', 'OZ'),
    ('Z', ';OZ'),
    ('Z', ''),
    ('O', 'a1'),
    ('O', '!(Y)[S]0'),
    ('1', '=2'),
    ('1', '<a[S]0'),
    ('2', '(E8'),
    ('2', 'a3'),
    ('3', '[S]0'),
    ('3', ''),
    ('0', '[S]'),
    ('0', ''),
    ('Y', 'a7'),
    ('7', '=a'),
    ('7', '<a'),
    ('Y', '!(Y)'),
    ('E', '(E8'),
    ('8', '+E'),
    ('8', '*E'),
    ('E', 'a')
]

```

```

class MP:
    TABLE = {
        'S': {
            'a': RULES[0],
            '!' : RULES[0]
        },
        'Z': {
            ';' : RULES[1],
            '[' : RULES[2],

```

```

        '↓': RULES[2],
    },
    '0': {
        'a': RULES[3],
        '!': RULES[4]
    },
    '1': {
        '=': RULES[5],
        '<': RULES[6],
    },
    '2': {
        '(': RULES[7],
        'a': RULES[8],
    },
    '3': {
        ';': RULES[10],
        '[': RULES[9],
        ']': RULES[10],
        '↓': RULES[10],
    },
    '0': {
        ';': RULES[12],
        '[': RULES[11],
        ']': RULES[12],
        '↓': RULES[12],
    },
    'Y': {
        'a': RULES[13],
        '!': RULES[16]
    },
    '7': {
        '=': RULES[14],
        '<': RULES[15]
    },
    'E': {
        '(': RULES[17],
        'a': RULES[20]
    },
    '8': {
        '+': RULES[18],
        '*': RULES[19]
    }
}

def _print_rule(self, rule: (str, str)):
    rule_num = RULES.index(rule) + 1
    print(f'{" " if rule_num < 10 else ""}{rule_num}. {rule[0]} -> {rule[1] if rule[1]
else "e"}')

def check(self, chain: str):
    stack = ['↓', 'S']
    chain += '↓'
    indexInSym = 0
    run_flag = True
    while run_flag:
        X = stack[-1]
        if (X in G_TERMINALS) or (X == '↓'):
            if X == chain[indexInSym]:
                stack.pop()
                indexInSym += 1
            else:
                raise Exception
        else:
            try:
                rule: (str, str) = MP.TABLE[X][chain[indexInSym]]
                stack.pop()
                stack += rule[1][::-1]

```

```

        self._print_rule(rule)
    except:
        raise Exception

run_flag = (X != '↓')

```

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

```

1. S -> OS`
4. O -> aO1
6. 1 -> =O2
9. 2 -> aO3
10. 3 -> [S]O`
1. S -> OS`
4. O -> aO1
6. 1 -> =O2
8. 2 -> (EE`
21. E -> a
19. 8 -> +E)
21. E -> a
3. Z -> e
13. 0 -> e
2. Z -> ;OS`
5. O -> !(Y) [S]O`
14. Y -> aY`
16. 7 -> <a
1. S -> OS`
4. O -> aO1
6. 1 -> =O2
8. 2 -> (EE`
21. E -> a
20. 8 -> *E)
21. E -> a
3. Z -> e
12. 0 -> [S]
1. S -> OS`
4. O -> aO1
6. 1 -> =O2
9. 2 -> aO3
11. 3 -> e
3. Z -> e
3. Z -> e
correct

1. S -> OS`
4. O -> aO1
6. 1 -> =O2
9. 2 -> aO3
11. 3 -> e
3. Z -> e
correct

1. S -> OS`
4. O -> aO1
6. 1 -> =O2
9. 2 -> aO3
10. 3 -> [S]O`
incorrect

1. S -> OS`
5. O -> !(Y) [S]O`
14. Y -> aY`
16. 7 -> <a
incorrect

```

Вывод: в ходе работы изучены и применены нисходящие методы обработки формальных языков.