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

Устраним самолеворекурсивное правило S → S;O:

S → OS′
S′ → ;OS′
S′ → ϵ
O → Y[S]
O → Y[S][S]
O → a=E
Y → a=a
Y → a<a
Y → !(Y)
E → (E+E)
E → (E*E)
E → a

Устраним левую рекурсию:

S → OS′
S′ → ;OS′
S′ → ϵ
O → Y[S]
O → Y[S][S]
O → a=E
Y → a=a
Y → a<a
Y → !(Y)
E → (E+E)
E → (E*E)
E → a

Имеются правила с идентичными префиксами, факторизуем эти правила:
S → OS′
S′ → ;OS′
S′ → ϵ
O → Y[S]O`
O` → [S]
O′ → ϵ
O → a=E
Y → aY`
Y` → =a
Y` → <a
Y → !(Y)
E → (EE`
E` → +E)
E` → *E)
E → a

Пересечение множеств ВЫБОР(O → Y[S]O`) и ВЫБОР(O → a=E) даёт непустое множество, устраним это недоразумение:
S → OS′
S′ → ;OS′
S′ → ϵ
O → aO1
O → !(Y)[S]O`
O1 → Y`[S]O`
O1 → =E
O` → [S]
O′ → ϵ
Y → aY`
Y` → =a
Y` → <a
Y → !(Y)
E → (EE`
E` → +E)
E` → *E)
E → a

Пересечение множеств ВЫБОР(O1 → Y`[S]O`) и ВЫБОР(O1 → =E) даёт непустое множество, факторизуем эти правила:
S → OS′
S′ → ;OS′
S′ → ϵ
O → aO1
O → !(Y)[S]O`
O1 → =O2
O1 → <a[S]O`
O2 → E
O2 → a[S]O`
O` → [S]
O′ → ϵ
Y → aY`
Y` → =a
Y` → <a
Y → !(Y)
E → (EE`
E` → +E)
E` → *E)
E → a

Пересечение множеств ВЫБОР(O2 → E) и ВЫБОР(O2 → a[S]O`) даёт непустое множество, факторизуем эти правила и получим LL(1) грамматика:
S → OS′
S′ → ;OS′
S′ → ϵ
O → aO1
O → !(Y)[S]O`
O1 → =O2
O1 → <a[S]O`
O2 → (EE`
O2 → aO3
O3 → [S]O`
O3 → ϵ
O` → [S]
O′ → ϵ
Y → aY`
Y` → =a
Y` → <a
Y → !(Y)
E → (EE`
E` → +E)
E` → *E)
E → 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 → OS′	a !
S′ → ;OS′	;
S′ → ϵ	┤]
O → aO1	a
O → !(Y)[S]O`	!
O1 → =O2	=
O1 → <a[S]O`	<
O2 → (EE`	(
O2 → aO3	a
O3 → [S]O`	[
O3 → ϵ	; ┤]
O` → [S]	[
O′ → ϵ	; ┤]
Y → aY`	a
Y` → =a	=
Y` → <a	<
Y → !(Y)	!

E → (EE`	(
E` → +E)	+
E` → *E)	*
E → a	a

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

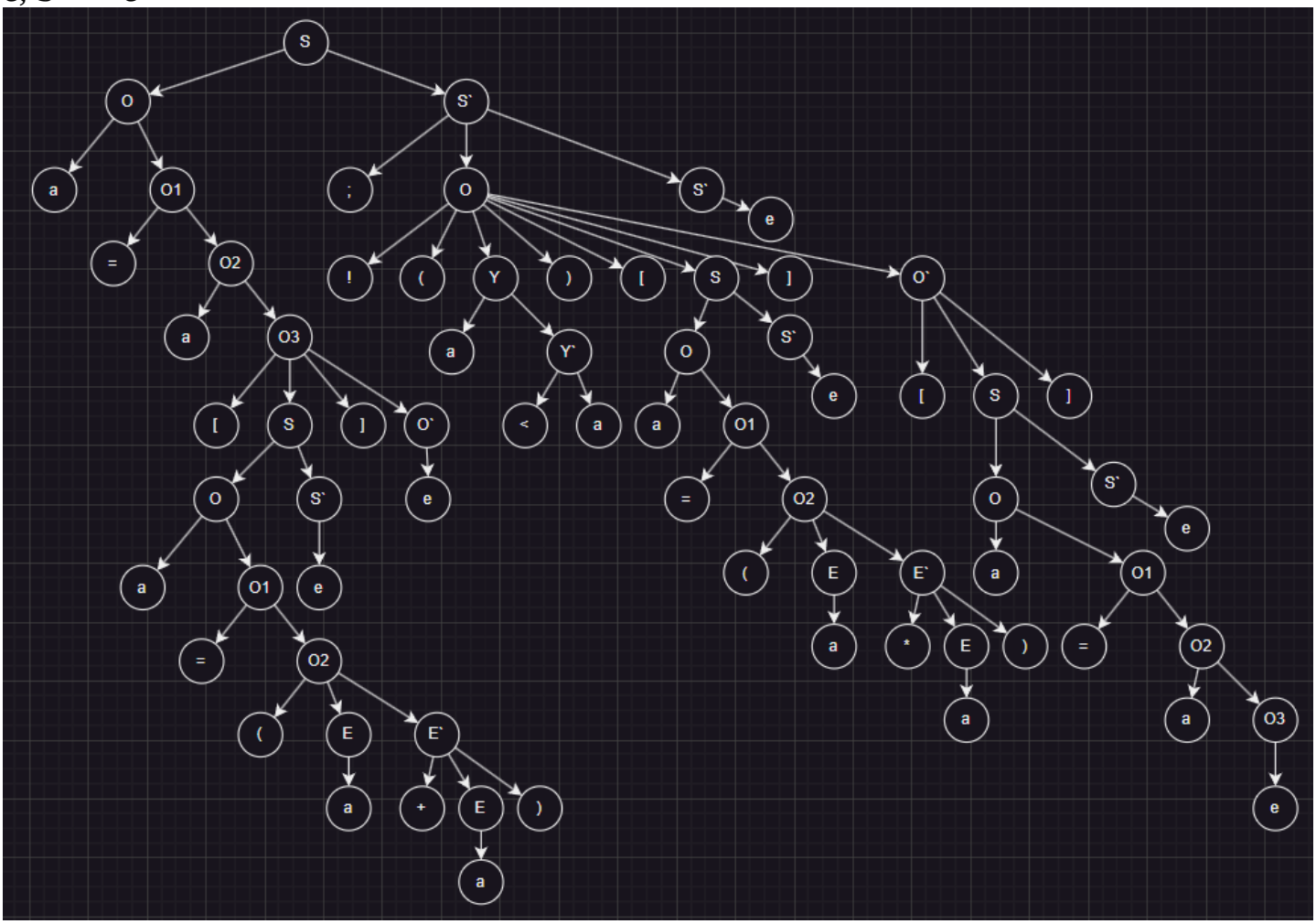
```
G_TERMINALS = ";[]a=<!()+*"

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]0'
        }
        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
```

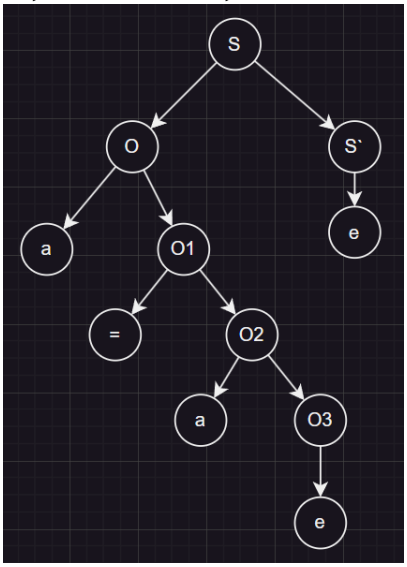
```
    def Yt(self):
        CHANGE = {
            '=': '=a',
            '<': '<a'
        }
        try:
            in_sym = self.chain[0]
            print(f'Y` => {CHANGE[in_sym]}')
            self.parse(CHANGE[in_sym])
        except Exception:
            self.err_flag = True

    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 => 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



Допустимая цепочка: a=a
Левый вывод: S => OS`; O => aO1; O1 => =O2; O2 => aO3; O3 => e; S` => 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 → OS′ a !
2. S′ → ;OS′ ;
3. S′ → ϵ ┤]
4. O → aO1 a
5. O → !(Y)[S]O` !
6. O1 → =O2 =
7. O1 → <a[S]O` <
8. O2 → (EE` (
9. O2 → aO3 a
10. O3 → [S]O` [
11. O3 → ϵ ; ┤]
12. O` → [S] [
13. O′ → ϵ ; ┤]
14. Y → aY` a
15. Y` → =a =
16. Y` → <a <
17. Y → !(Y) !
18. E → (EE` (
19. E` → +E) +
20. E` → *E) *
21. E → a a

МП-распознаватель:

	;	[	]	a	=	<	!	(	)	+	*	┤
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	
[		выт сд										
]			выт сд									
a				выт сд								
(								выт сд				
)									выт 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(a), сдвиг
#16 ЗАМЕНИТЬ(a), 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],
```

```
        },
        'O': {
            '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)):
        print(f'{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.

```
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
incorrect
```

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