Филипенков В. А.
ИУ7-22М

Отчет по **ЛР№ 1** по дисциплине
«Конструирование компиляторов»

РАСПОЗНАВАНИЕ ЦЕПОЧЕК РЕГУЛЯРНОГО ЯЗЫКА

Вариант 3.

1. Задание

Напишите программу, которая в качестве входа принимает произвольное регулярное выражение, и выполняет следующие преобразования:

1) По регулярному выражению строит НКА.
2) По НКА строит эквивалентный ему ДКА.
3) По ДКА строит эквивалентный ему КА, имеющий наименьшее возможное количество состояний по алгоритму Хопкрофта.
4) Моделирует минимальный КА для входной цепочки из терминалов исходной грамматики.

2. Текст программы (алгоритмов)

1)

```
operations = {
    '*': star,
    '+': plus,
    '|': alternate,
    ',': concatenate
}
priorities = {
    '|': 0,
    ',': 1,
    '*': 2,
    '+': 2
}

binary = ['|', ',']
unary = ['*', '+', '?']
```

```
def prepare_regexp(regexp):
    if len(regexp) == 0:
        return ''
    new = []
    last = None
    for c in regexp:
```

```
        if last is None:
            last = c
            new.append(c)
            continue
        if last in unary and c == '(' \
            or last in unary and is_character(c) \
            or is_character(last) and is_character(c) \
            or last == ')' and is_character(c) \
            or is_character(last) and c == '(':
            new.append(',')
        new.append(c)
        last = c
    print(new)
    return ''.join(new)
```

```
def merge_tables(A, B):
    keys = set(list(A.table.keys()) + list(B.table.keys()))
    new_final = [state + A.num_of_states() for state in B.final_states]
    new_final.extend(A.final_states)
    new_table = {}
    for k in keys:
        new_row = []
        if k in A.table:
            new_row.extend(A.table[k])
        else:
            new_row.extend([[] for _ in range(A.num_of_states())])
        if k in B.table:
            new_row.extend([[s + A.num_of_states() for s in states] for
states in B.table[k]])
        else:
            new_row.extend([[] for _ in range(B.num_of_states())])
        new_table[k] = new_row
    return NDA(table=new_table, final_states=new_final)


def concatenate(A, B):
    merged = merge_tables(A, B)
    for start in A.final_states:
        merged.add_transition(start, EPSILON, A.num_of_states())
    merged.final_states = [s + A.num_of_states() for s in B.final_states]
    return merged
```

```
def alternate(A, B):
    merged = merge_tables(A, B)
    shifted_finals = [f + 1 for f in merged.final_states]
    shifted_table = {}
    for char, state_list in merged.table.items():
        shifted_table[char] = [[]] + [[state + 1 for state in states] for
states in state_list] + [[]]
    new = NDA(table=shifted_table, final_states=[])
    new.add_transition(0, EPSILON, 1)
    new.add_transition(0, EPSILON, A.num_of_states() + 1)
    for f in shifted_finals:
        new.add_transition(f, EPSILON, new.num_of_states() - 1)
    new.final_states = [new.num_of_states() - 1]
    return new


def star(A):
    shifted_finals = [f + 1 for f in A.final_states]
    shifted_table = {}
    for char, state_list in A.table.items():
        shifted_table[char] = [[]] + [[state + 1 for state in states] for
states in state_list] + [[]]
    new = NDA(table=shifted_table, final_states=[])
    for f in shifted_finals:
        new.add_transition(f, EPSILON, 1)
        new.add_transition(f, EPSILON, new.num_of_states() - 1)
    new.add_transition(0, EPSILON, 1)
    new.add_transition(0, EPSILON, new.num_of_states() - 1)
    new.final_states = [new.num_of_states() - 1]
    return new


def plus(A):
    return concatenate(A, star(A))


def primitive_nfda(actual_string):
    table: Dict[str, List[List[int]]] = {}
    for i, c in enumerate(actual_string):
        if c not in table:
            table[c] = [[] for _ in range(len(actual_string) + 1)]
        table[c][i].append(i + 1)
    return NDA(table=table, final_states=[len(actual_string)])
```

```
def create_nfda(regexp):
    op_stack = []
    automata_stack = []
    buffer = ''

    def avalanche(priority=-1):
        while len(op_stack) != 0 \
                and op_stack[-1] != '(' \
                and (op_stack[-1] not in operations.keys() or
priorities[op_stack[-1]] > priority):
            op = op_stack[-1]
            if op in binary:
                automata_stack.append(operations[op](automata_stack[-2],
automata_stack[-1]))
                automata_stack.pop(-2)
                automata_stack.pop(-2)
                op_stack.pop()
            elif op in unary:
                automata_stack.append(operations[op](automata_stack[-1]))
                automata_stack.pop(-2)
                op_stack.pop()
        if priority == -1 and len(op_stack) != 0 and op_stack[-1] == '(':
            op_stack.pop()

    regexp = prepare_regexp(regexp)

    for c in regexp:
        if c in list(operations.keys()) + ['(', ')']:
            if buffer != '':
                automata_stack.append(primitive_nfda(buffer))
            buffer = ''
        if c in operations:
            if len(op_stack) == 0 or op_stack[-1] in ['(', ')'] or
priorities[op_stack[-1]] < priorities[c]:
                op_stack.append(c)
            else:
                avalanche(priorities[c])
                op_stack.append(c)
        elif c == '(':
            op_stack.append('(')
        elif c == ')':
            avalanche()
        else:
            buffer += c
```

```
    if buffer != '':
        automata_stack.append(primitive_nfda(buffer))
    avalanche()

    return automata_stack[-1]
```

2)

```
def convert_to_fda(nfda):
    links = []
    newStates = [set(nfda.eps_close(0))]
    visitedStates = []
    alphabet = [x for x in list(nfda.table.keys()) if x != EPSILON]
    while len(newStates) > 0:
        tmp = newStates.pop()
        if tmp in visitedStates:
            continue
        visitedStates.append(tmp)
        for char in alphabet:
            newTmp = set(nfda.forward(tmp, char))
            if len(newTmp) != 0:
                newStates.append(newTmp)
                links.append((tmp, char, newTmp))
    formatted_links = []
    for link in links:
        formatted_links.append((visitedStates.index(link[0]), link[1],
visitedStates.index(link[2])))
    links = formatted_links
    old_final = set(nfda.final_states)
    new_final = [i for i, s in enumerate(visitedStates) if
s.intersection(old_final)]
    new_table = {}
    for k in alphabet:
        new_table[k] = [None for _ in enumerate(visitedStates)]
    for link in links:
        new_table[link[1]][link[0]] = link[2]
    return DA(table=new_table, final_states=new_final)
```

3)

```
def minimize_fda(fda):
    def split_set(target, splitter, split_char):
        R1 = set()
        R2 = set()
        for v in target:
```

```
            if fda.table[split_char][v] in splitter:
                R1.add(v)
            else:
                R2.add(v)
        return R1, R2

    sets = [{*fda.final_states}]
    non_final =
{*list(range(fda.num_of_states()))}.difference(fda.final_states)
    if len(non_final) > 0:
        sets.append(non_final)
    queue = []
    for c in fda.alphabet():
        for s in sets:
            queue.append((s, c))
    while len(queue) > 0:
        splitter, char = queue.pop(0)
        for s in sets:
            R1, R2 = split_set(s, splitter, char)
            if len(R1) > 0 and len(R2) > 0:
                sets.remove(s)
                sets.extend([R1, R2])
                if (s, char) in queue:
                    queue.remove((s, char))
                    queue.append((R1, char))
                    queue.append((R2, char))
                else:
                    if len(R1) < len(R2):
                        queue.append((R1, char))
                    else:
                        queue.append((R2, char))

    first_state_index = [sets.index(s) for s in sets if 0 in s][0]
    first_state = sets.pop(first_state_index)
    sets.insert(0, first_state)

    num_of_states = len(sets)
    new_table = {k: [None] * num_of_states for k in fda.alphabet()}
    for i, s in enumerate(sets):
        for v in s:
            for c in fda.alphabet():
                new_indexes = [sets.index(s) for s in sets if
fda.table[c][v] in s]
```

```
                new_table[c][i] = None if len(new_indexes) == 0 else
new_indexes[0]
    new_final = [sets.index(s) for s in sets if
s.intersection(fda.final_states)]
    return DA(table=new_table, final_states=new_final)
```

4)

```
    def model_check(self, check_str):
        check_arr = [*check_str]
        size = len(self.table[list(self.table.keys())[0]])
        Ssize = len(self.table)
        true_table = np.full((size,size,Ssize), None)
        j = 0
        for char, state_list in self.table.items():
            for i, s in enumerate(state_list):
                if s != None:
                    true_table[i][s][j] = char
            j += 1

        carette = 0

        while(True):
            if not check_arr:
                break
            for i in range(size):
                if check_arr:
                    arr = []
                    for a in true_table[carette]:
                        for b in a:
                            arr.append(b)
                    if check_arr[0] not in arr:
                            print('ERROR')
                            return
                    for symbol in true_table[carette][i]:
                        if check_arr[0] == symbol:
                            check_arr.pop(0)
                            carette = i
                            break

        if not check_arr and carette in self.final_states:
            print('OK')
        else:
            print('ERROR')
        return
```

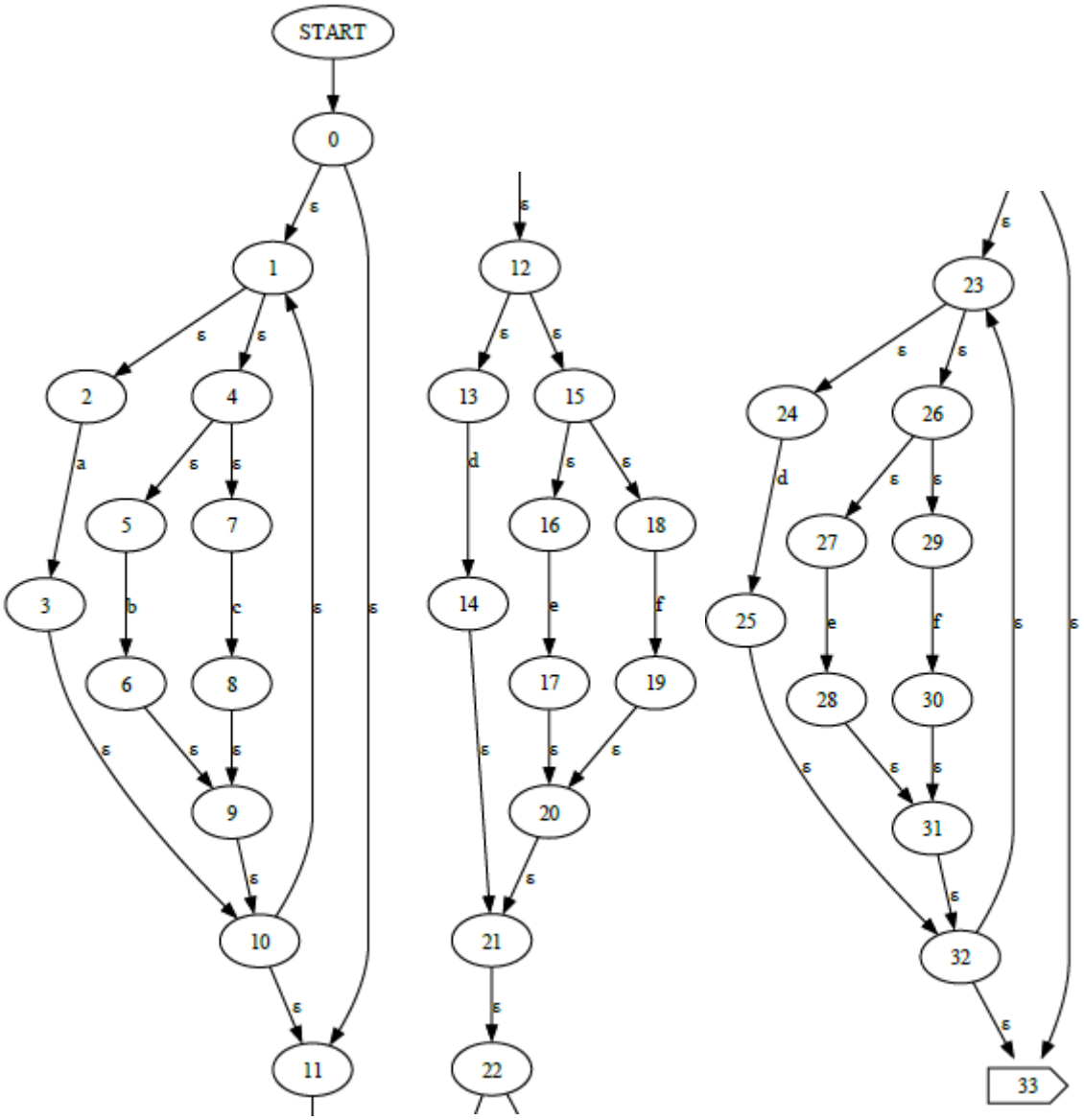
3. Тесты

Регулярное выражение	*Проверяемое выражение*	*Ожидаемый результат*
a*b*c*	abc	OK
	bc	OK
	ccccccc	OK
	aaaaabbbbbbbccccccc	OK
		OK
	bca	ERROR
	abca	ERROR
a+b+c	abc	OK
	aaaaaaabbbbbc	OK
	abca	ERROR
	ccccccc	ERROR
		ERROR
(a|b|c)*(d|e|f)*	abcdef	OK
	ad	OK
	aaaaeefff	OK
		OK
	ada	ERROR
((a|b|c)*|(d|e|f)*)*	abcdbdafedfbcfa	OK
		OK
	g	ERROR
((a|b)(a|b)(a|b))*	aba	OK
	bbbbbb	OK
	aaaaaabbb	OK
		OK
	ab	ERROR
	abab	ERROR
	bbaba	ERROR
	babaaab	ERROR

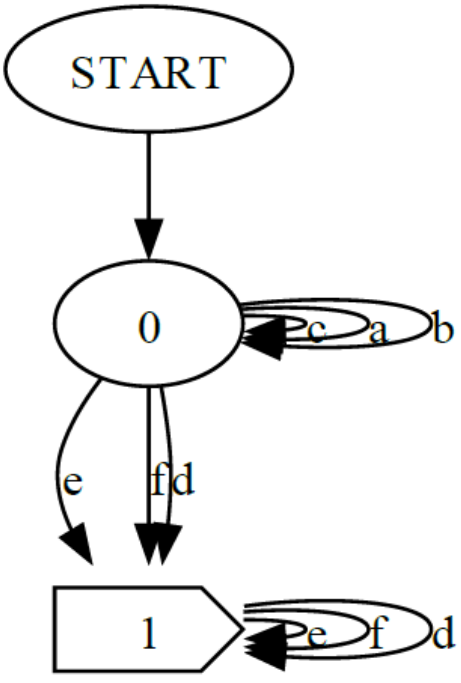
4. Результаты работы программы

```
C:\Users\user\Desktop\KK\lab1>python main.py
Введите регулярное выражение: (a|b|c)*(d|e|f)+
Введите строку для моделирования МКА (для выхода введите 'N'): abcdef
OK
Введите строку для моделирования МКА (для выхода введите 'N'): abcabcd
OK
Введите строку для моделирования МКА (для выхода введите 'N'): d
OK
Введите строку для моделирования МКА (для выхода введите 'N'):
ERROR
Введите строку для моделирования МКА (для выхода введите 'N'): ada
ERROR
Введите строку для моделирования МКА (для выхода введите 'N'): adbf
ERROR
Введите строку для моделирования МКА (для выхода введите 'N'): N
```

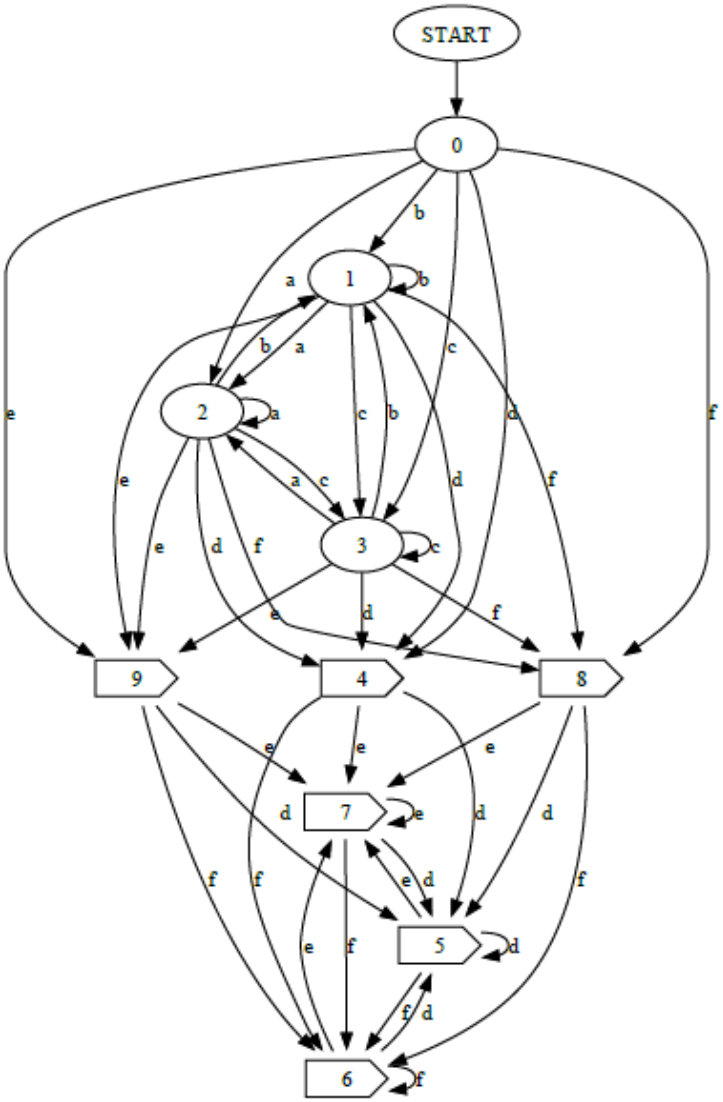
НКА


START
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
a
b
c
d
e
f
ε

МДКА


START
0
1
c
a
b
e
f
d
e
f
d

ДКА



5. *Ответы на контрольные вопросы*

1) Какие из следующих множеств регулярны? Для тех, которые регулярны, напишите регулярные выражения.

a. Множество цепочек с равным числом нулей и единиц (не регулярное множество).

b. Множество цепочек из {0, 1} * с четным числом нулей и нечетным числом единиц.

???????

c. Множество цепочек из {0, 1} *, длины которых делятся на 3.

((0|1)(0|1)(0|1))*

d. Множество цепочек из {0, 1} *, не содержащих подцепочки 101.

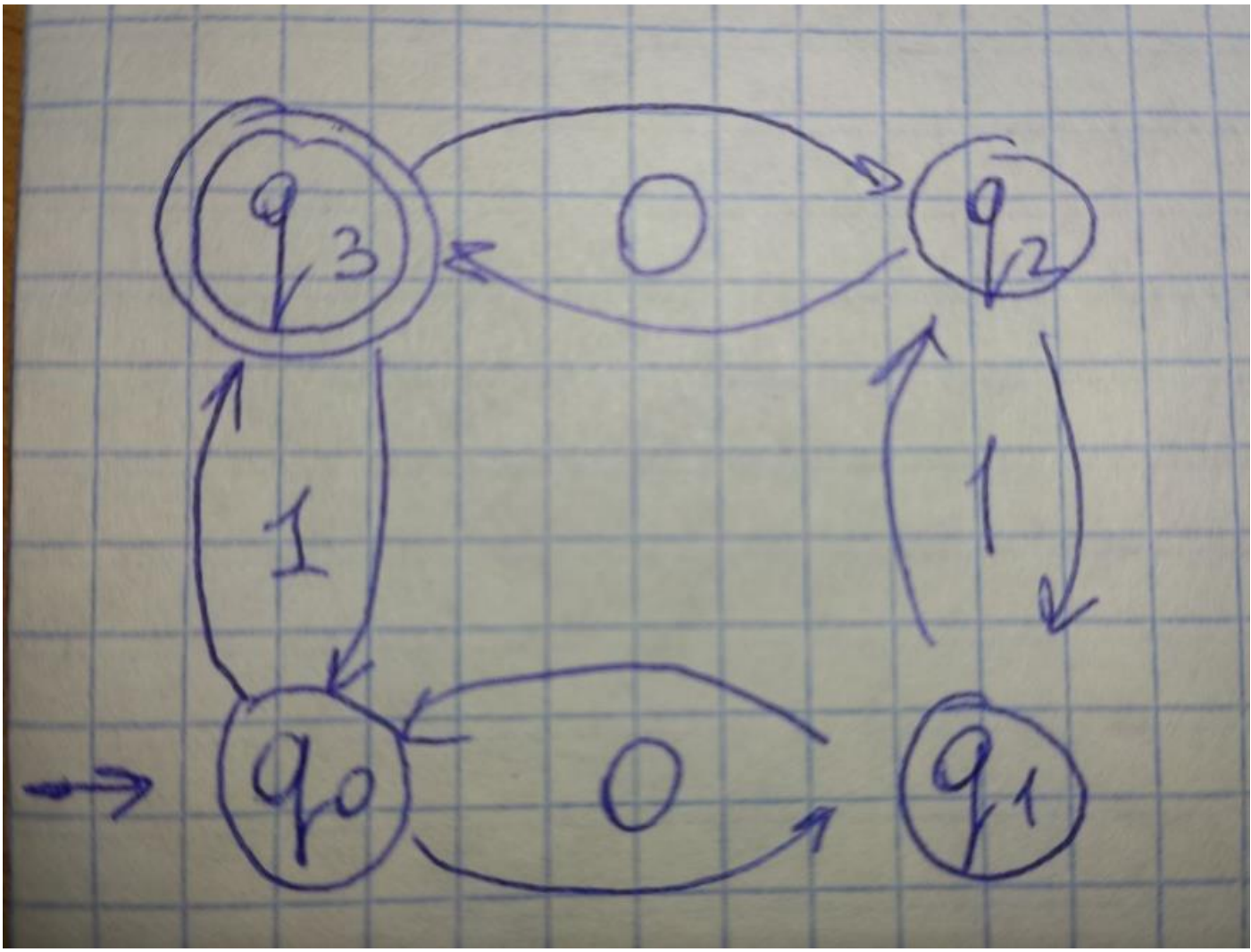
0*(1|00|000)*0*

2) Найдите праволинейные грамматики для тех множеств из вопроса 1, которые регулярны.

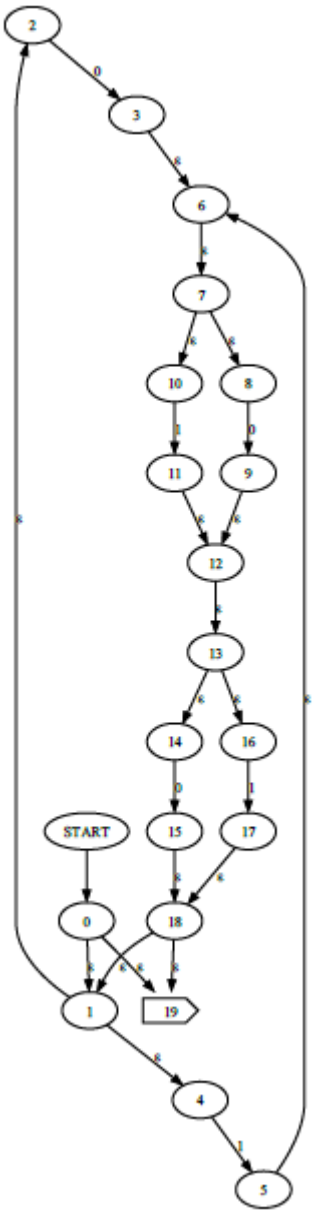
b	c	d
S → 0A S → 1C A → 0S A → 1B B → 1A B → 0C C → 0B C → 1S C → ε	S → A A → 0B A → 1B A → ε B → 0C B → 1C C → 0A C → 1A	S → A A → 0A A → B B → 1B B → 00B B → 000B B → C C → 0C C → ε

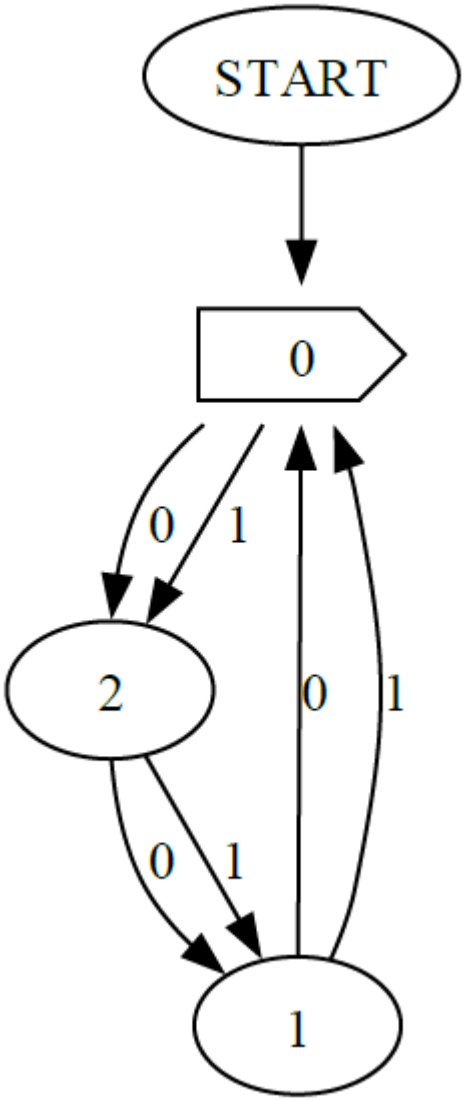
3) Найдите детерминированные и недетерминированные конечные автоматы для тех множеств из вопроса 1, которые регулярны.

b) *???????*

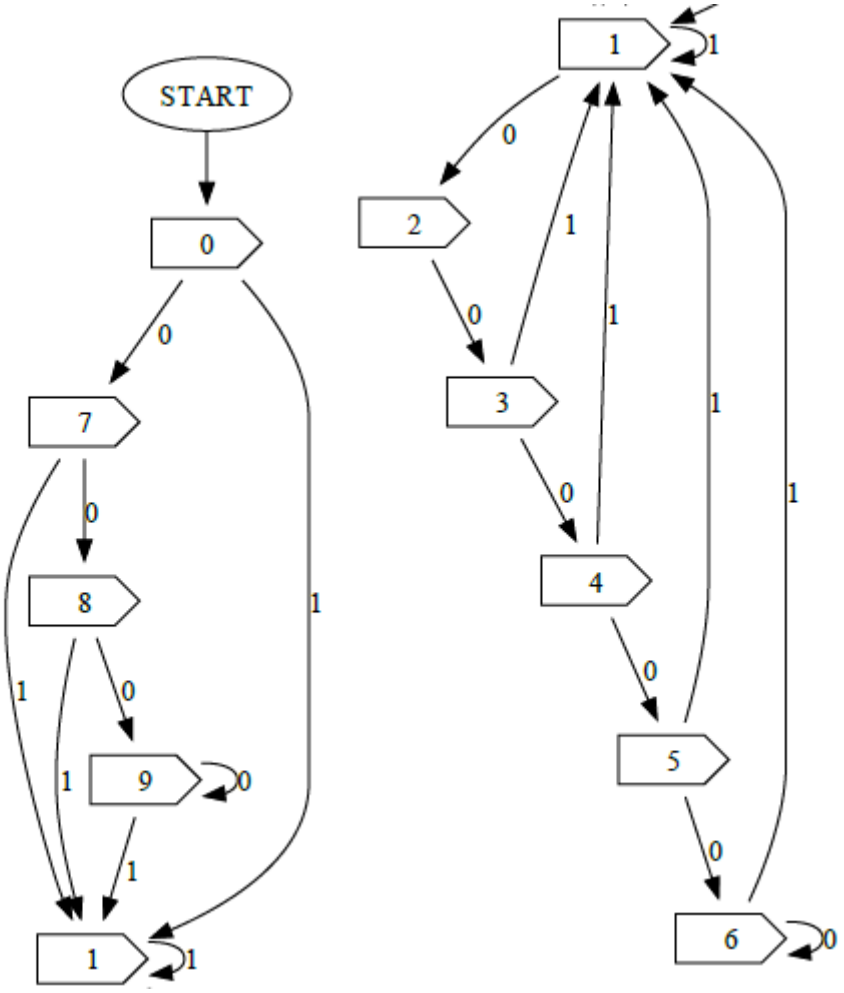


c) *((0|1)(0|1)(0|1))**



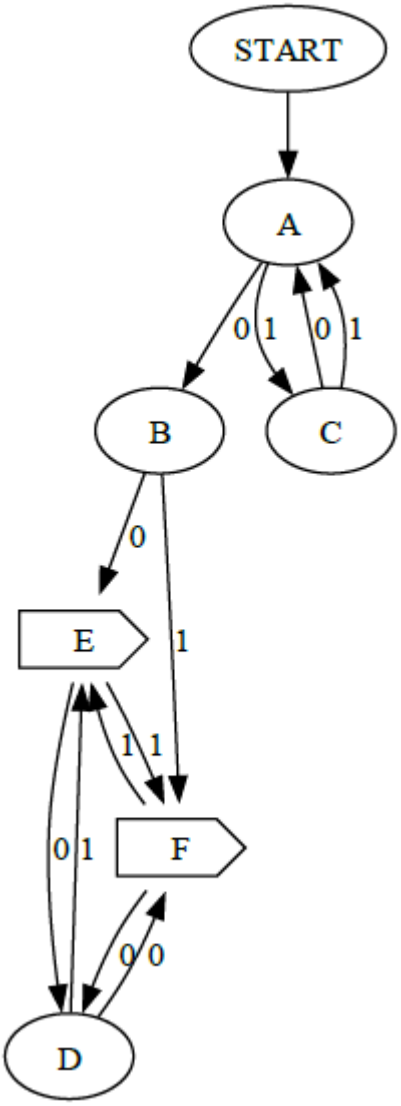


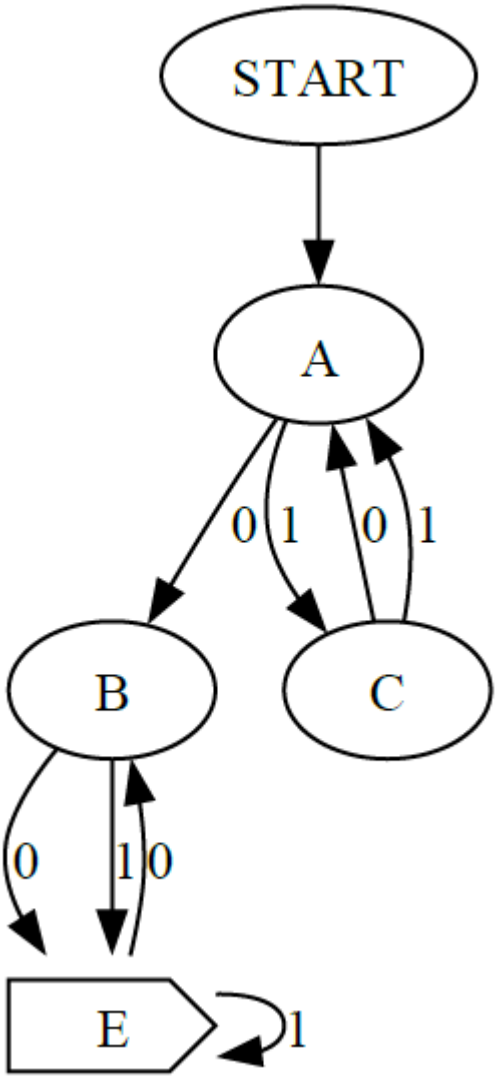
d) *0*(1|00|000)*0**



4) Найдите конечный автомат с минимальным числом состояний для языка, определяемого автоматом M = ({A, B, C, D, E}, {0, 1}, d, A, {E, F}), где функция задается таблицей

Состояние	Вход 0	1
A	B	C
B	E	F
C	A	A
D	F	E
E	D	F
F	D	E


START
A
0
1
0
1
B
C
0
E
1
1
1
0
1
F
0
0
D


START
A
0
1
0
1
B
C
0
1
0
E
1