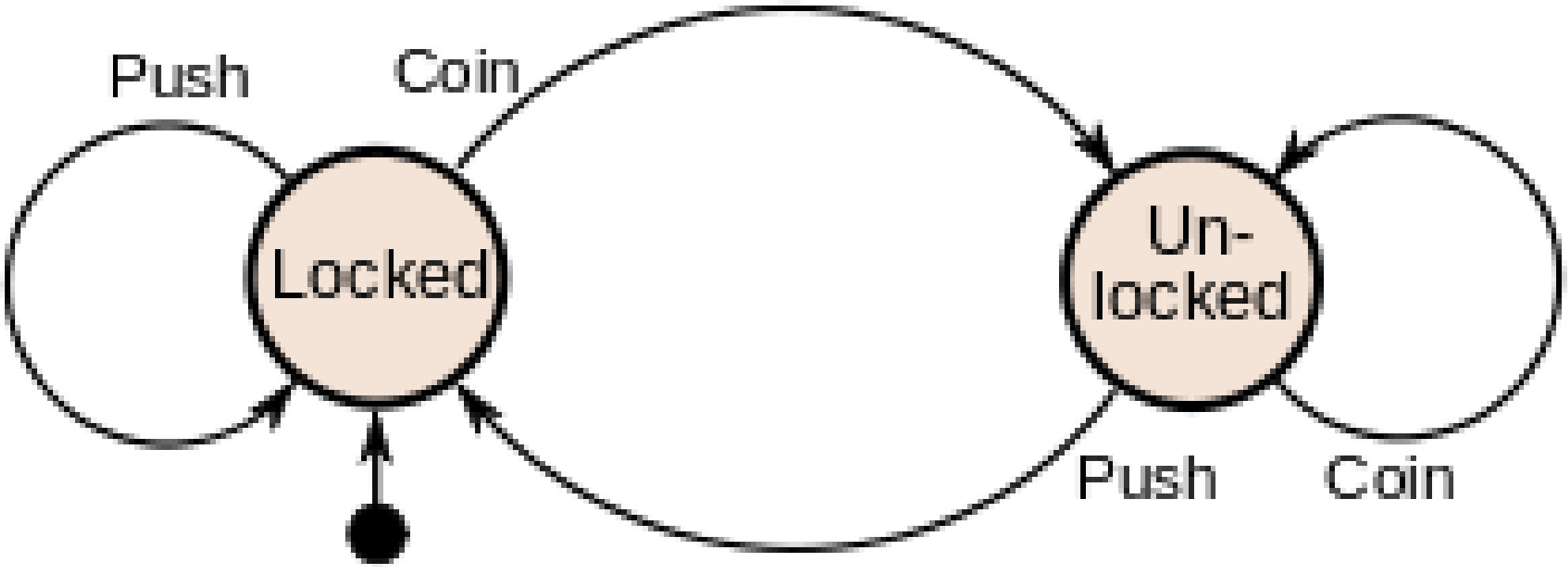
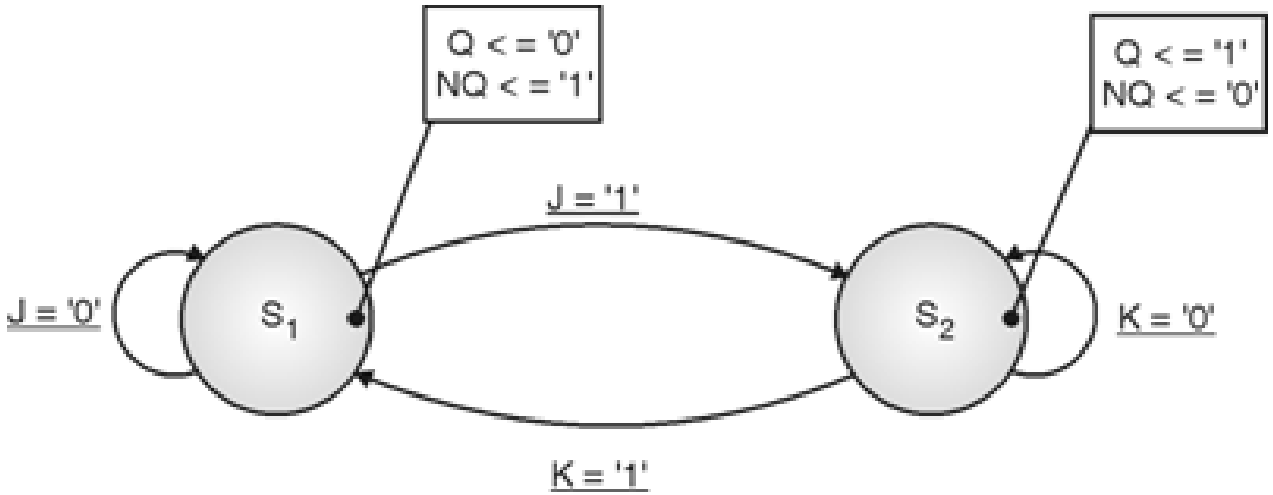
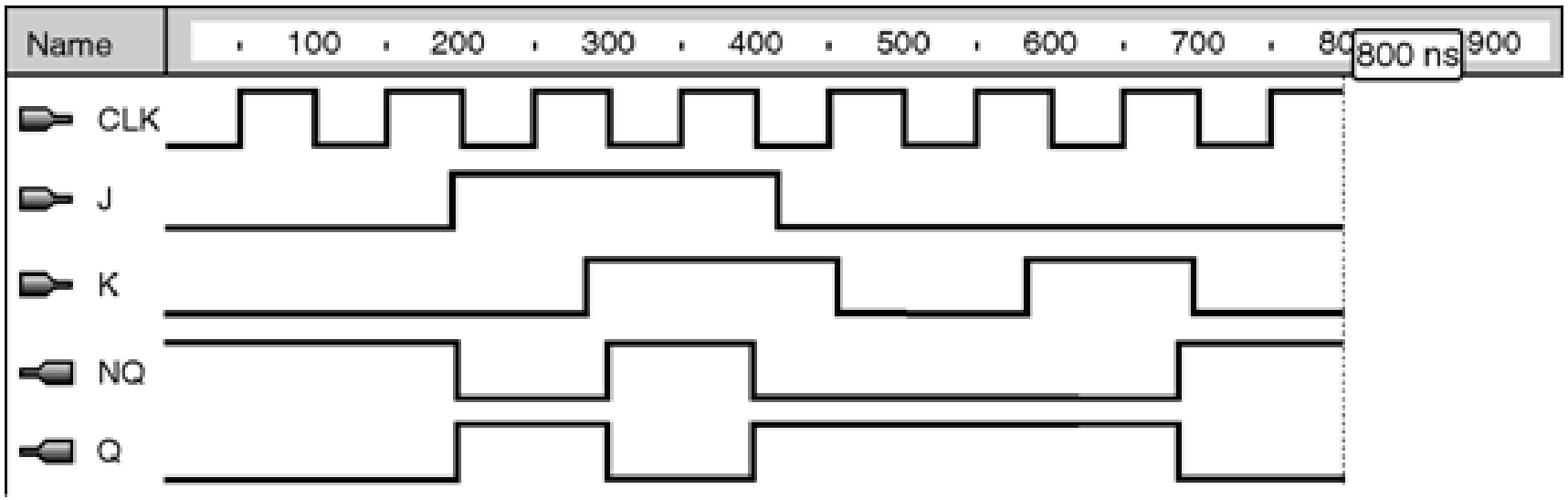
Finite state machine (or finite-state automaton)

A FSM is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. Finite state machines can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics.


Push
Coin
Locked
Un-
locked
Push
Coin


J
CLK
K
Q
NQ


Q < = '0'
NQ < = '1'
Q < = '1'
NQ < = '0'
J = '1'
J = '0'
S1
S2
K = '0'
K = '1'


Name
100
200
300
400
500
600
700
800
800 ns
900
CLK
J
K
NQ
Q

Рекламная интеграция

Таблица истинности для дешифратора АЛУ приведена в **Табл. 7.2**. Значения всех трех битов сигнала *ALUControl* были приведены в **Табл. 5.1**. Так как сигнал *ALUOp* никогда не равен 11, то для упрощения логики мы можем использовать неопределенные значения X1 и 1X вместо 01 и 10.

Когда сигнал *ALUOp* равен 00 или 01, АЛУ должно складывать или вычитать соответственно. Когда он равен 10, то значение *ALUControl* должно определяться полем `funct`. Заметьте, что для команд типа *R*, которые мы добавили, первые два бита поля `funct` всегда равны 10, так что мы можем их проигнорировать для упрощения дешифратора.

Управляющие сигналы для всех команд уже были описаны, когда мы создавали тракт данных. Таблица истинности для основного дешифратора, показывающая зависимость управляющих сигналов от значения `opcode`, приведена в **Табл. 7.3**.

Табл. 7.2. Таблица истинности дешифратора АЛУ

ALUOp	`funct`	*ALUControl*
00	X	010 (сложение)
X1	X	110 (вычитание)
1X	100000 (`add`)	010 (сложение)
1X	100010 (`sub`)	110 (вычитание)
1X	100100 (`and`)	000 (логическое «И»)
1X	100101 (`or`)	001 (логическое «ИЛИ»)
1X	101010 (`slt`)	111 (установить, если меньше)

Для всех команд типа *R* основной дешифратор формирует одинаковые сигналы; эти команды отличаются только сигналами, сформированными дешифратором АЛУ. Для команд, которые не пишут в регистровый файл (например, `sw` или `beq`), управляющие сигналы *RegDst* и *MemtoReg* могут принимать любое состояние, то есть являются неопределенными (X); адрес и данные, приходящие на порт записи регистрового файла, не имеют никакого значения, так как *RegWrite* равен нулю.

Для создания дешифратора вы можете использовать любой известный вам метод синтеза комбинационных схем.

Табл. 7.3. Таблица истинности основного дешифратора

Команда	*Opcode*	*RegWrite*	*RegDst*	*ALUSrc*	*Branch*	*MemWrite*	*MemtoReg*	*ALUOp*
Команды типа *R*	000000	1	1	0	0	0	0	10
`lw`	100011	1	0	1	0	0	1	00
`sw`	101011	0	X	1	0	1	X	00
`beq`	000100	0	X	0	1	0	X	01

Пример 7.1. ФУНКЦИОНИРОВАНИЕ ОДНОТАКТНОГО ПРОЦЕССОРА

Определите значения управляющих сигналов, а также части тракта данных, которые задействованы при выполнении команды `or`.

Решение: на **Рис. 7.13** показаны управляющие сигналы и пути движения данных во время выполнения команды `or`. Счетчик команд указывает на ячейку памяти, из которой выбирается команда.

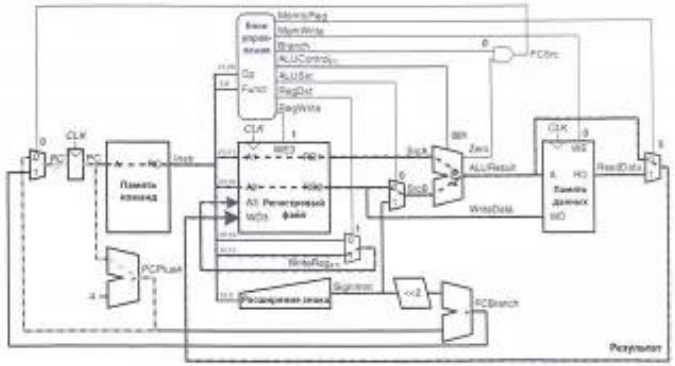


Рис. 7.13. Управляющие сигналы и пути движения данных при выполнении команды `or`

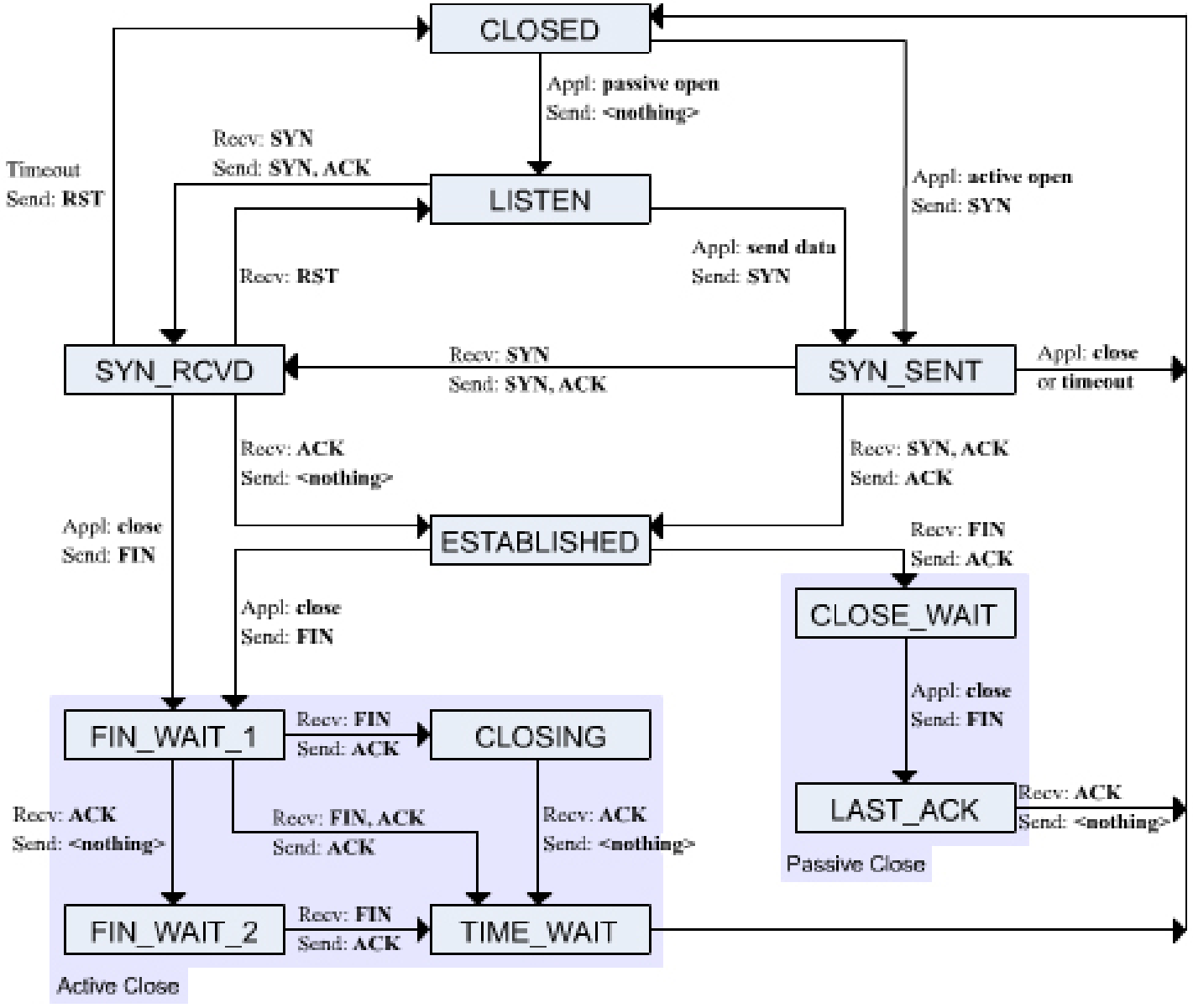
Прохождение данных через регистровый файл и АЛУ показано синей пунктирной линией. Из регистрового файла читаются два операнда; их номера регистров задаются полями $Instr_{25:21}$ и $Instr_{20:16}$. На вход *SrcB* нужно подать значение, прочитанное из второго порта регистрового файла, а не *SignImm*, так что *ALUSrc* должен быть равен нулю. Команда `or` – это команда типа *R*, то есть *ALUOp* равен 10, поэтому значение сигнала *ALUControl*, которое для команды `or` должно быть равно 001, будет вычислено на основе поля `funct`. Сигнал *Result* формируется в АЛУ, поэтому *MemtoReg* должен быть равен нулю. Результат записывается в регистровый файл, поэтому *RegWrite* будет равен единице. Команда ничего не пишет в память, так что сигнал *MemWrite* равен нулю.

Запись в регистр результата также показана синей пунктирной линией. Номер этого регистра задается в поле `rd` ($Instr_{15:11}$), так что сигнал *RegDst* должен быть равен единице.

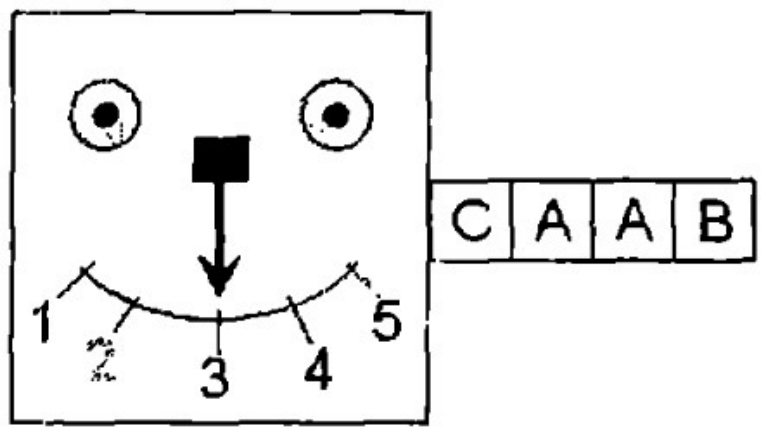
Запись нового значения в счетчик команд показана серой пунктирной линией. Так как команда `or` не является командой условного перехода, то сигнал Branch равен нулю и, соответственно, *PCSrc* тоже равен нулю. В результате счетчик команд получает новое значение из *PCPlus4*.

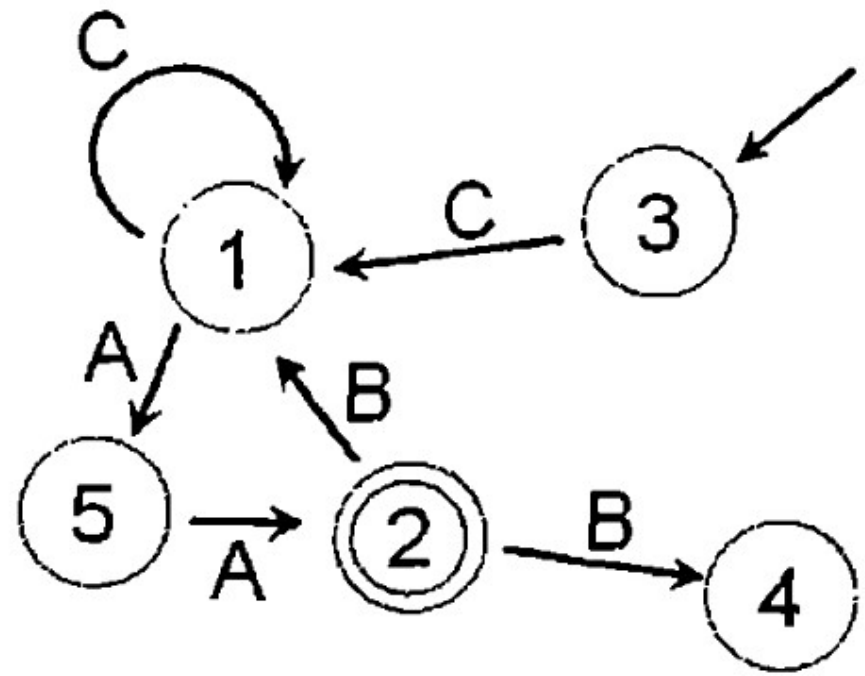
Важно иметь в виду, что по цепям, не отмеченным пунктиром, тоже передаются какие-то сигналы и данные, однако для этой конкретной команды совер-

TCP


CLOSED
Appl: passive open
Send: <nothing>
Recv: SYN
Send: SYN, ACK
Timeout
Send: RST
LISTEN
Appl: active open
Send: SYN
Appl: send data
Send: SYN
Recv: RST
SYN_RCVD
Recv: SYN
Send: SYN, ACK
SYN_SENT
Appl: close
or timeout
Recv: ACK
Send: <nothing>
Recv: SYN, ACK
Send: ACK
Appl: close
Send: FIN
ESTABLISHED
Recv: FIN
Send: ACK
CLOSE_WAIT
Appl: close
Send: FIN
FIN_WAIT_1
Recv: FIN
Send: ACK
CLOSING
Appl: close
Send: FIN
LAST_ACK
Recv: ACK
Send: <nothing>
Passive Close
Recv: ACK
Send: <nothing>
Recv: FIN, ACK
Send: ACK
Recv: ACK
Send: <nothing>
FIN_WAIT_2
Recv: FIN
Send: ACK
TIME_WAIT
Active Close

2, B → 4
3, C → 1
5, A → 2
1, A → 5
2, B → 1
1, C → 1





Formal definition of the Deterministic Finite Automata

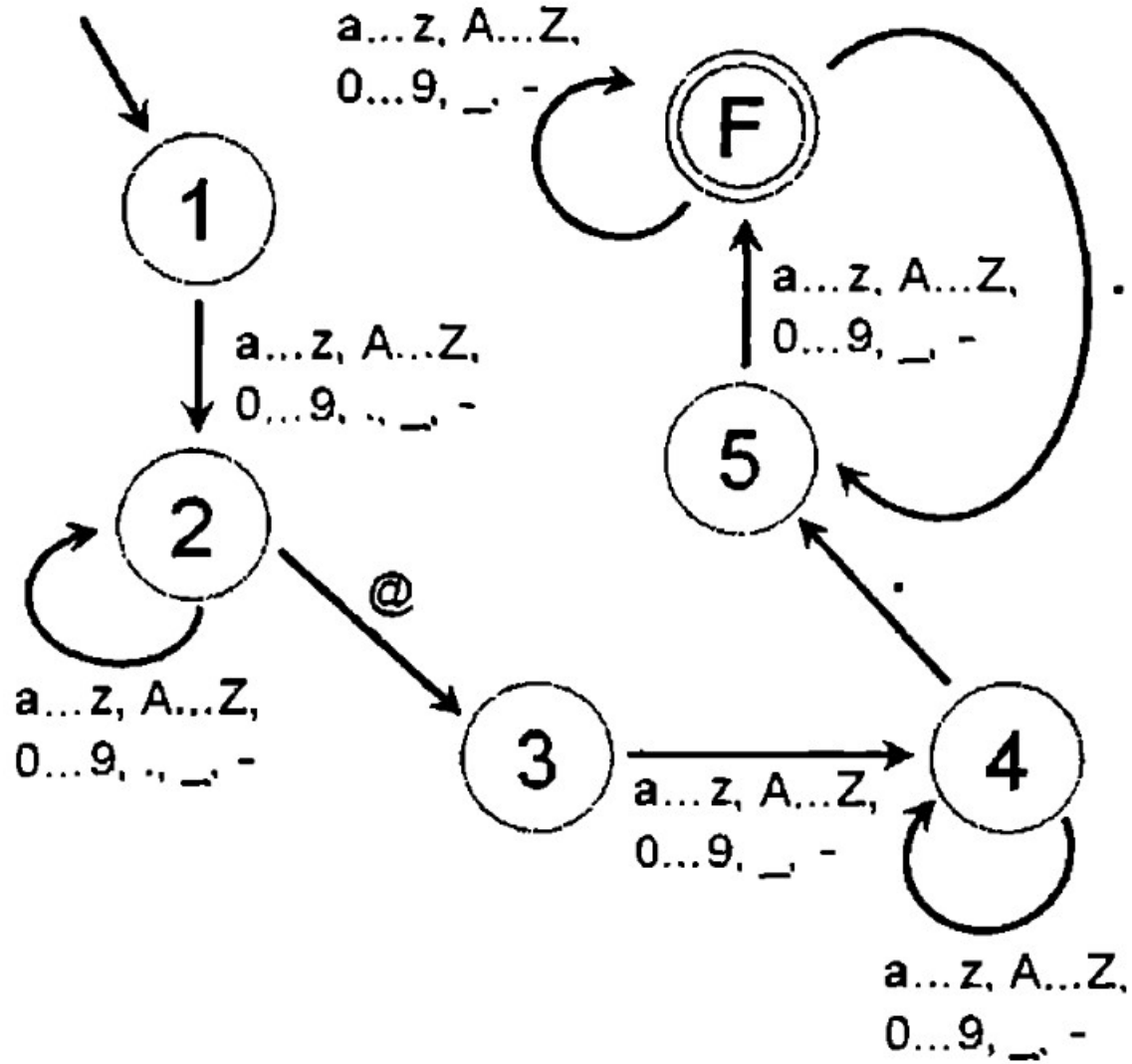
A deterministic finite automaton (DFA) is described by a five-element tuple: (Q, Σ, δ, q0, F)

Q = a finite set of states
Σ = a finite, nonempty input alphabet
δ = a series of transition functions
q0 = the starting state
F = the set of accepting states

There must be exactly one transition function for every input symbol

```
[a-zA-Z0-9._-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+
```

```
[a-zA-Z0-9._-]+@[a-zA-Z0-9_-]+(\.[a-zA-Z0-9_-]+)+
```

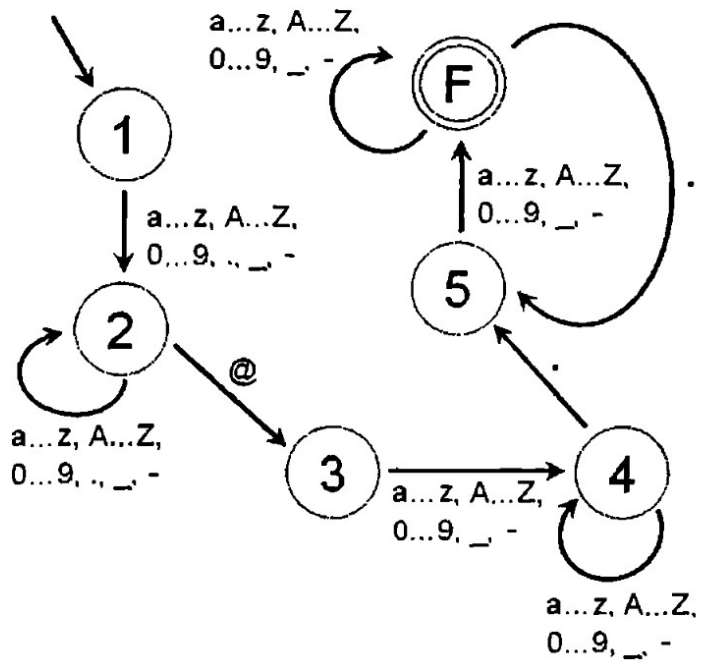


zeus@olympus

zeus@olympus.gov

```
    enum StateType {S1, S2, ...,  Sn};
    enum SymbolType {C1, C2, ..., Cm, Cend};

    StateType state = StateType.S1;
    SymbolType symbol;
    try
    {
        while((symbol = СчитатьСледующийСимволЛенты()) !=
                        SymbolType.Cend)
        {
            switch(state)

        {
         case StateType.S1:
              обработка правил вида (S1, symbol -> Sk)
              если правило не найдено, throw new Exception();
         case StateType.S2:
              обработка правил вида (S2, symbol -> Sk)
              если правило не найдено, throw new Exception();
            ...
         case StateType.Sn:
              обработка правил вида (Sn, symbol -> Sk)
              если правило не найдено, throw new Exception();
         }
    }

    предпринимаем какие-то действия в зависимости от того,
    является ли state допускающим состоянием
}
catch(Exception)
{
    вывод сообщения об ошибке «недопустимый входной символ»
}
```

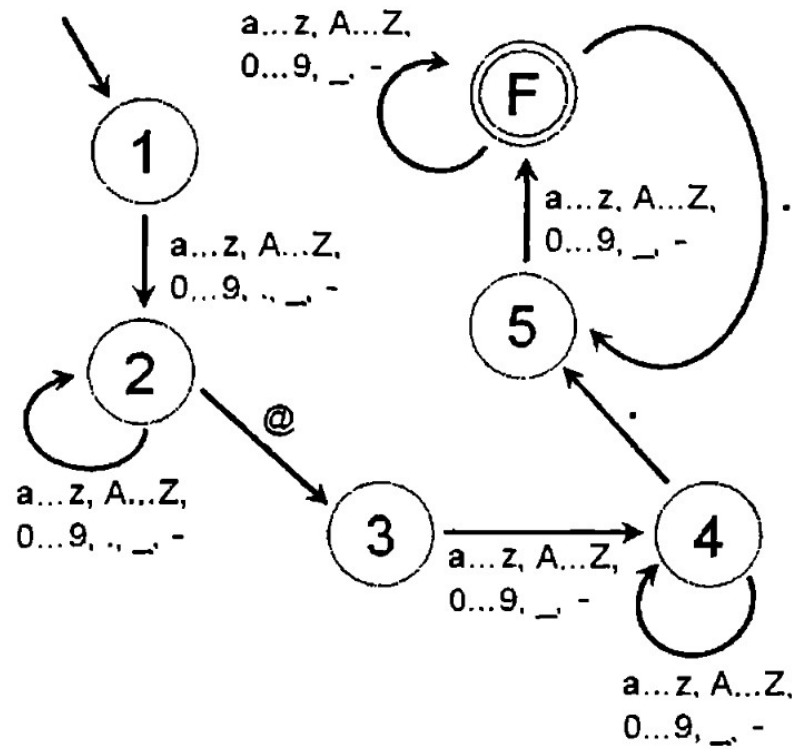
```
static bool InMainRange(char c) // в диапазоне [a-zA-Z0-9_-]
{
    return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||
           (c >= '0' && c <= '9') || (c == '_' || c == '-');
}

static void Main(string[] args)
{
    string s = Console.ReadLine() + '\0';
                            // символ '\0' – признак конца
    char state = '1';       // начальное состояние
    int i = 0;              // индекс текущего символа
    char symbol;            // текущий символ
```



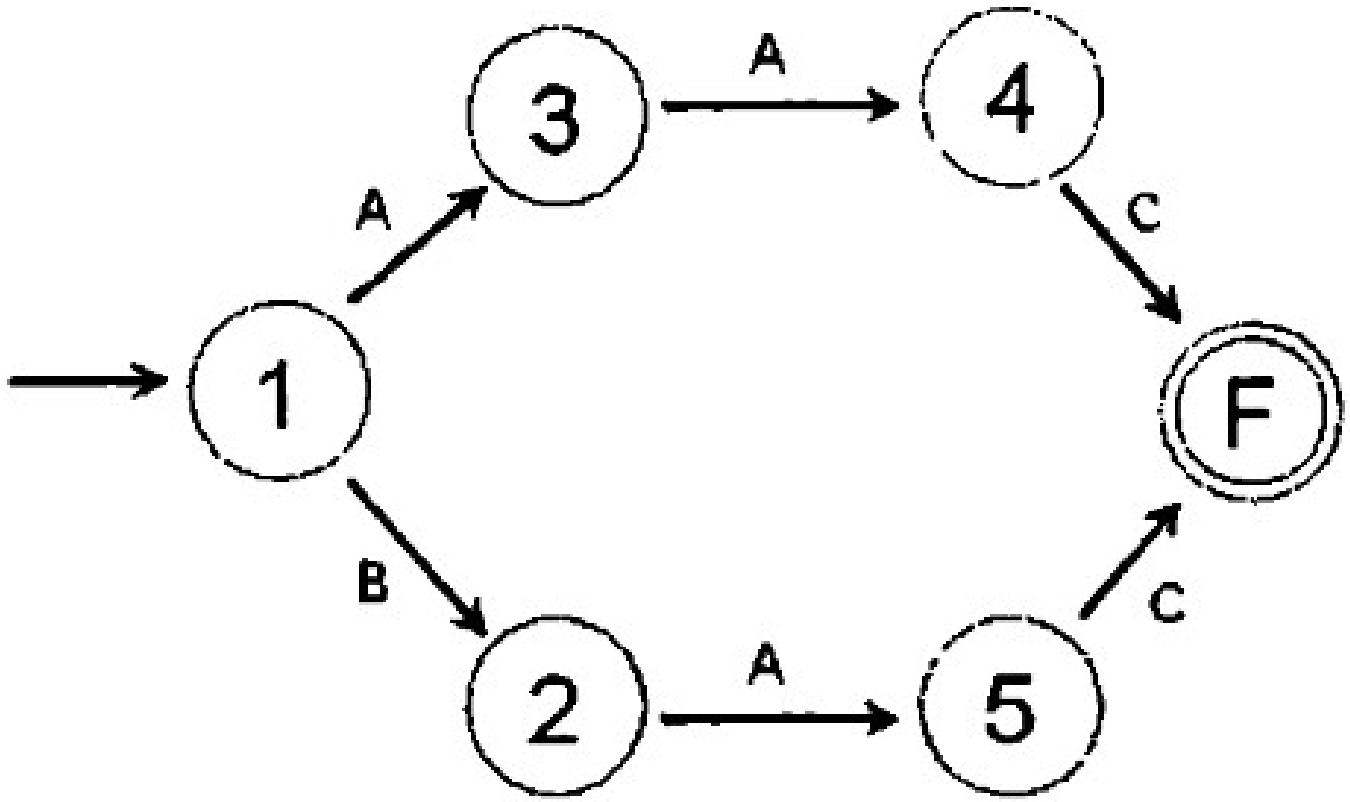

```
try
{
    while((symbol = s[i++]) != '\0')
                            // пока «лента» не кончилась
    {
        switch(state)       // обычная реализация
                            // конечного автомата
        {
           case '1': if(InMainRange(symbol) || symbol == '.')
                            state = '2';
                     else
                            throw new Exception();
                     break;

           case '2': if(InMainRange(symbol) || symbol == '.')
                            state = '2';
                     else if(symbol == '@')
                            state = '3';
                     else
                            throw new Exception();
                     break;

           case '3': if(InMainRange(symbol))
                            state = '4';
                     else
                            throw new Exception();
                     break;
```

```
            case '4': if(InMainRange(symbol))
                          state = '4';
                      else if(symbol == '.')
                          state = '5';
                      else
                          throw new Exception();
                      break;

            case '5': if(InMainRange(symbol))
                          state = 'F';
                      else
                          throw new Exception();
                      break;

            case 'F': if(InMainRange(symbol))
                          state = 'F';
                      else if(symbol == '.')
                          state = '5';
                      else
                          throw new Exception();
                      break;
        }
    }
    // печатаем True или False в зависимости типа
    // состояния state
    Console.WriteLine(state == 'F');
}
```

```
    catch(Exception)    // «правило не найдено»
    {
        Console.WriteLine("обнаружен недопустимый символ");
    }
}
```

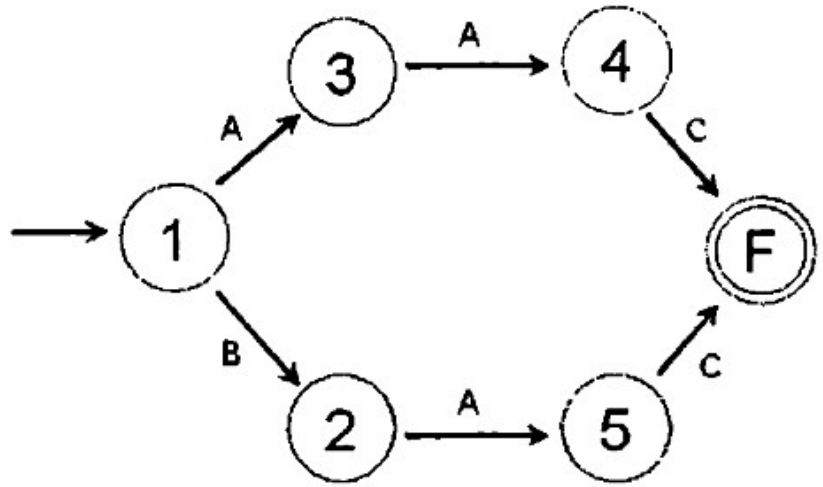


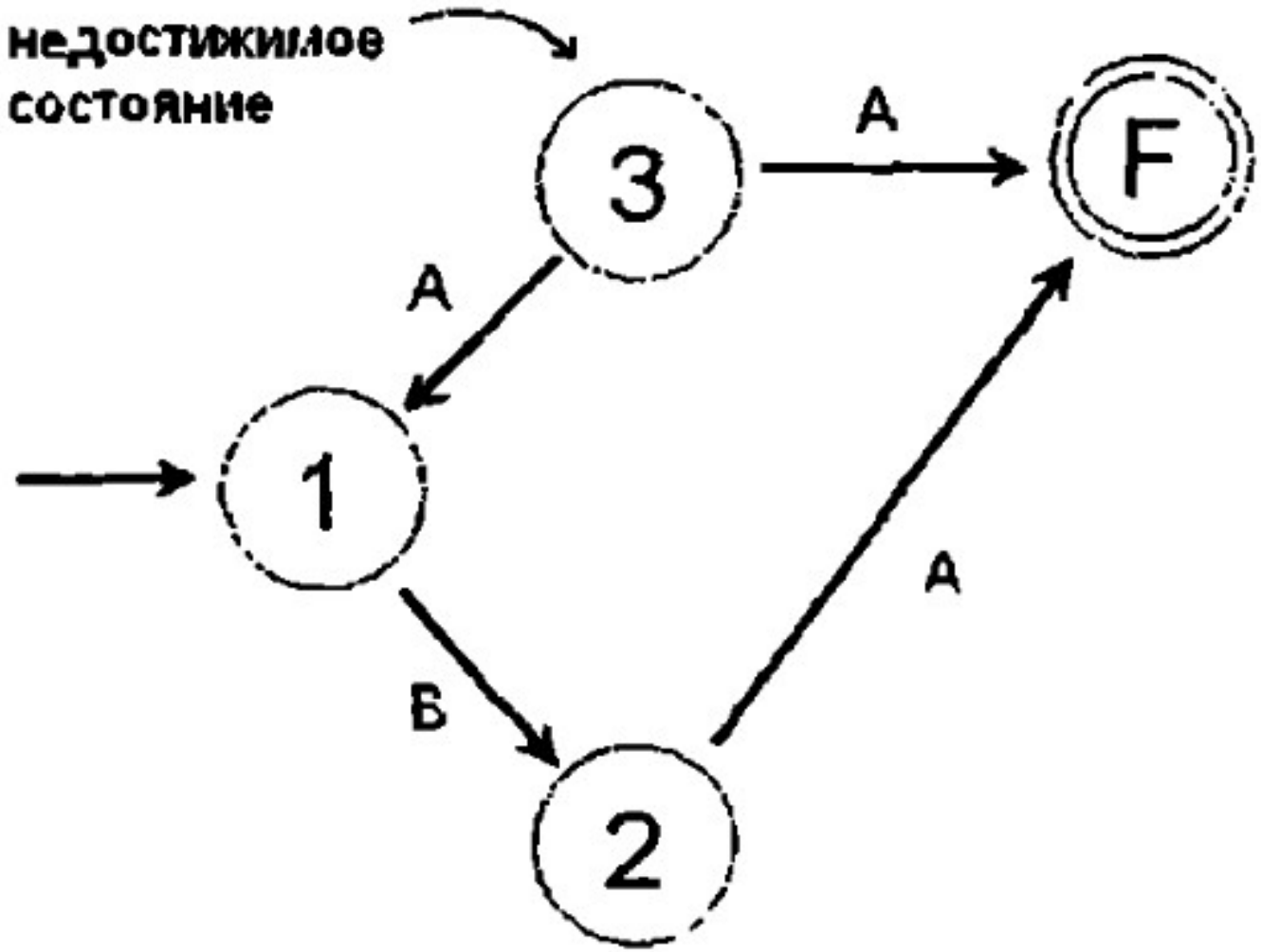
Эквивалентность состояний

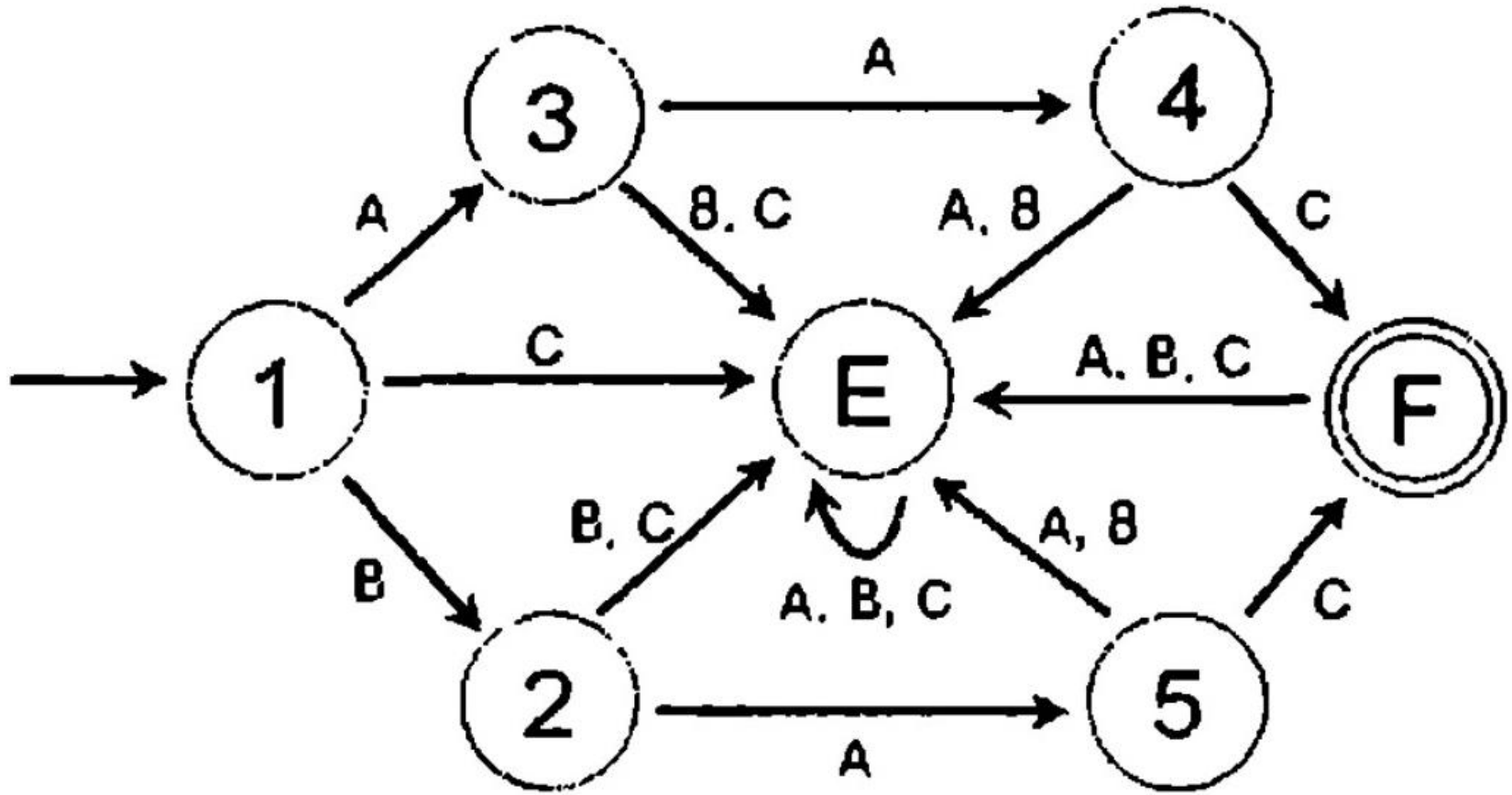


Минимизация автомата

```
удалить недостижимые состояния и связанные с ними правила;

создать таблицу всевозможных пар состояний вида (p, q);

ОТМЕТИТЬ те пары, где одно из состояний является допускающим,
а другое  –  нет;

DO
    found = false;
    ЕСЛИ существует неотмеченная пара (p, q), такая,
    что для некоторого элемента входного алфавита a
    пара (δ(p, a), δ(q, a)) отмечена
        ОТМЕТИТЬ пару (p, q);
        found = true;
WHILE found  // то есть пока изменения происходят

заменить каждое множество эквивалентных друг другу состояний
на единственное новое; соответствующим образом изменить
таблицу переходов;
```




недостижимое
состояние
3
A
F
A
1
A
Б
2


3
A
4
A
B, C
A, B
C
1
C
E
A, B, C
F
B, C
A, B
B
A, B, C
C
2
A
5

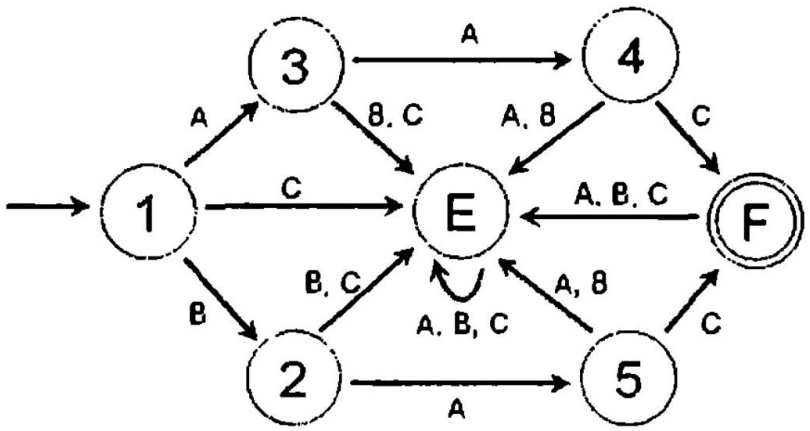
Минимизация автомата

удалить недостижимые состояния и связанные с ними правила;

создать таблицу всевозможных пар состояний вида (p, q);

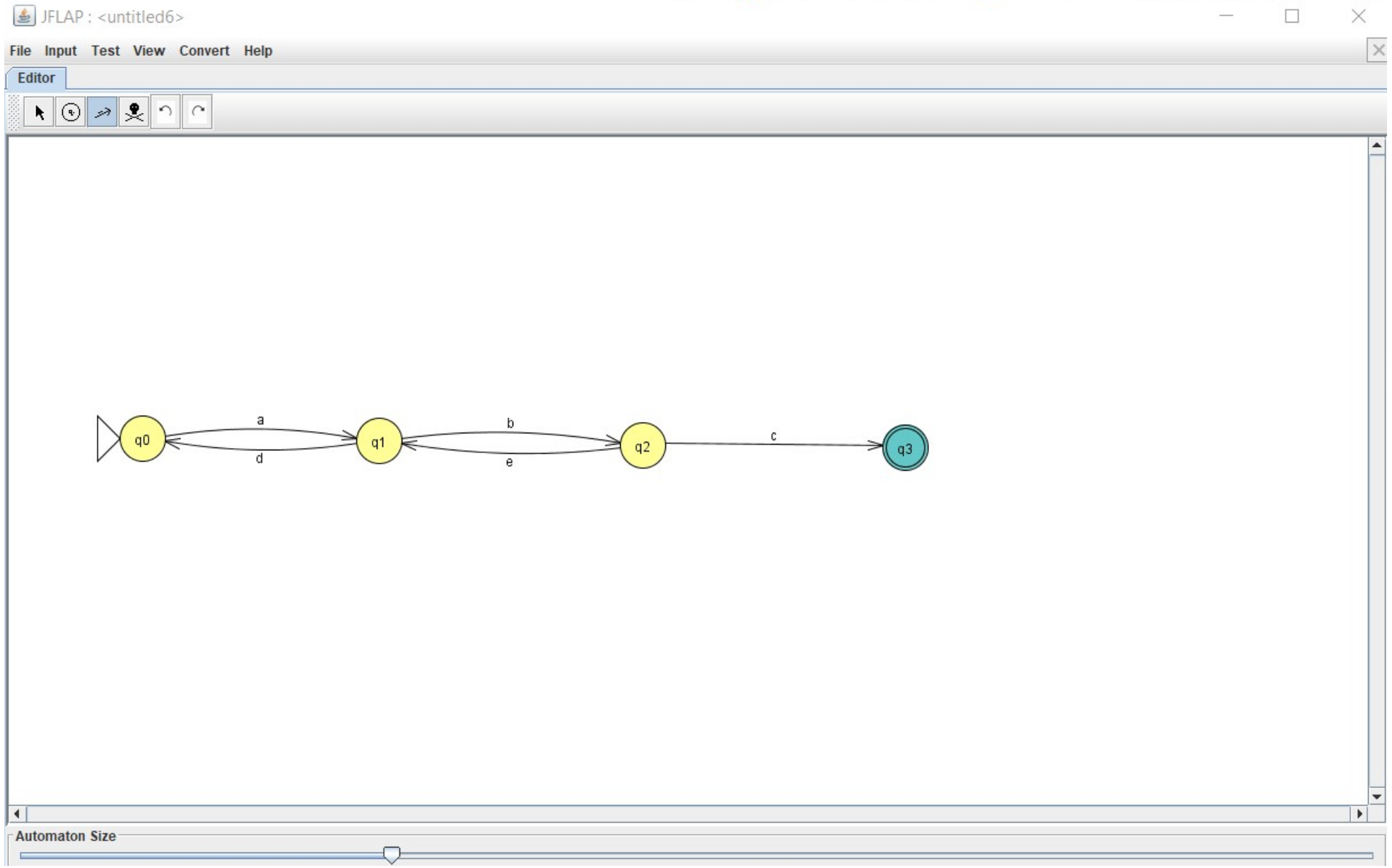
ОТМЕТИТЬ те пары, где одно из состояний является допускающим,
а другое — нет;

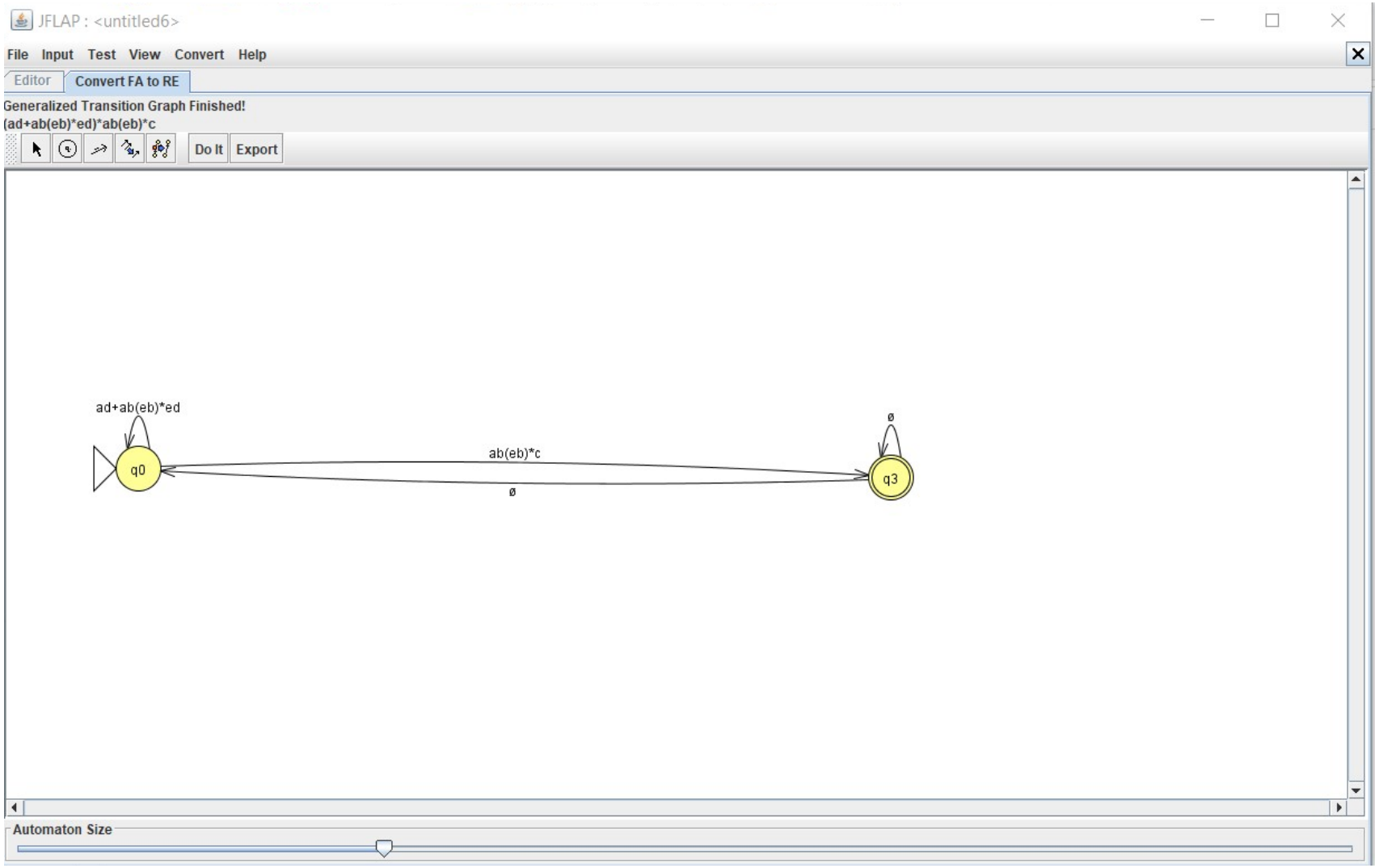
```
DO
    found = false;
    ЕСЛИ существует неотмеченная пара (p, q), такая,
    что для некоторого элемента входного алфавита a
    пара (δ(p, a), δ(q, a)) отмечена
        ОТМЕТИТЬ пару (p, q);
        found = true;
WHILE found  // то есть пока изменения происходят
```



заменить каждое множество эквивалентных друг другу состояний
на единственное новое; соответствующим образом изменить
таблицу переходов;

```
ЦИКЛ по всем состояниям q
     e_class[q] = q;
ЦИКЛ по всем неотмеченным парам (p, q)
    ЦИКЛ по всем состояниям s
        ЕСЛИ e_class[s] == p, ТО e_class[s] = q
```


JFLAP : <untitled6>
File Input Test View Convert Help
Editor
q0
q1
q2
q3
a
d
b
e
c
Automaton Size



$$(ad+ab(eb)*ed)*ab(eb)*c$$