СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

Тема 2.

Трансляция. Этапы трансляции. Лексический анализ.

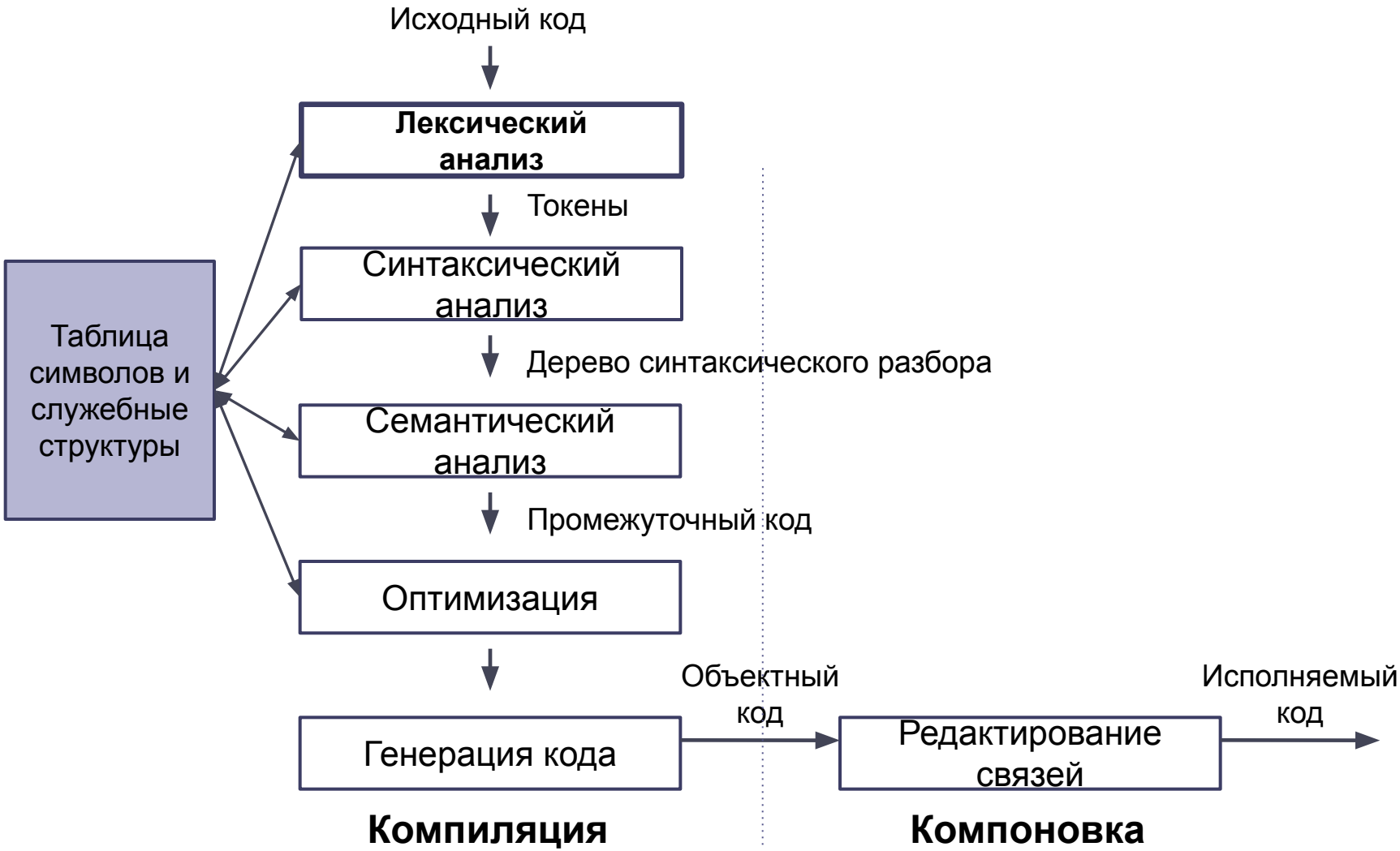
Трансляция

Трансляция (*англ*. **translation**) программы – преобразование программы, написанной на некотором языке программирования в форму, пригодную для ее исполнения вычислительной системой, или преобразование программы, представленной на одном языке программирования, в программу на другом языке в определённом смысле равносильную исходной.

Основные функции

- Компиляция
- Интерпретация
- Ассемблирование

Фазы трансляции



Лексический анализ

Лексический анализ (*англ*. **lexical analysis**, **scanning**) – процесс аналитического разбора входной последовательности символов (исходного кода) при котором из входной последовательности выделяются связные группы символов, называемые **лексемами**, которые затем классифицируются и преобразуются в пару «**токен**» -«**лексическое значение**».

```
y = x + 3;
```

(*ident*, ‘y’)
(*assgn*, --)
(*ident*, ‘x’)
(*add*, --)
(*intconst*, 3)

Функции лексического анализа

- удаление «пустых» символов и комментариев
- распознавание идентификаторов и ключевых слов
- распознавание констант
- распознавание разделителей и знаков операций

Регулярные выражения

Регулярное выражение (*англ.* **regular expression**) определяет множество строк над некоторым алфавитом Σ, дополненным символом ε (пустой символ).

```
alpha  → a | b | c | d | ... | z | A | B | C | D | ... | Z

digit  → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

identifier  → (alpha | _) (alpha | digit | _)*

intcon      → digit digit*

symbol      → + | - | * | / | ( | ) | :=

floatcon    → digit* . digit digit* ((E|e)(+|-|ε) digit digit*)|ε
```

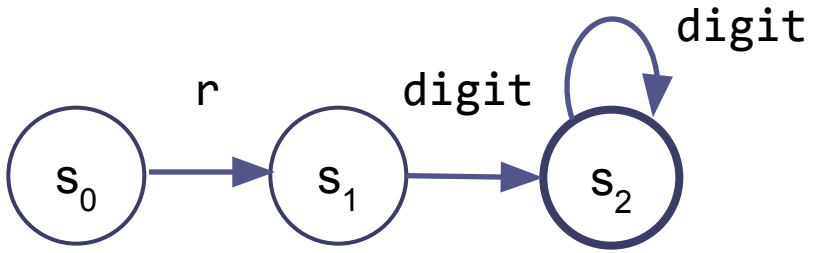
Операции, определенные для регулярных выражений

- Если $a \in \Sigma$, то a – регулярное выражение
- Если a и b – регулярные выражения, то
  - $a \mid b$ – регулярное выражение (альтернация)
    $L(a) \cup L(b) = \{s | s \in L(a)$ или $s \in L(b)\}$
  - ab – регулярное выражение (конкатенация)
    $L(a) \cdot L(b) = \{st | s \in L(a)$ и $t \in L(b)\}$
  - a^* – регулярное выражения (замыкание Клини)
    $L(a)^* = \bigcup_0^\infty L(a)$
  - (a) – регулярное выражение
- ε – регулярное выражение

Регулярные выражения и КА

Любое регулярное выражение *r* соответствует абстрактной машине, которая распознает язык *L(r)*. Такая абстрактная машина называется **конечным автоматом**.

```
r digit digit*
```

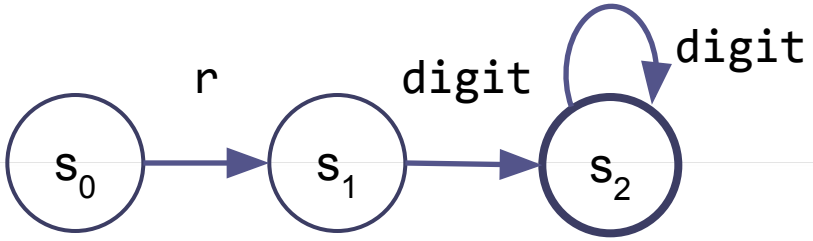


Формальное определение КА

Конечный автомат определяется кортежем $< Q, \Sigma, \delta, q_0, F >$, где

- $Q = \{q_i\}$ – множество состояний
- $\Sigma = \{\sigma_i\}$ – алфавит
- $\delta: Q \times \Sigma \to Q$ – функция переходов
- q_0 – начальное состояние
- $F = \{q_j\}$ – множество конечных (допускающих состояний)

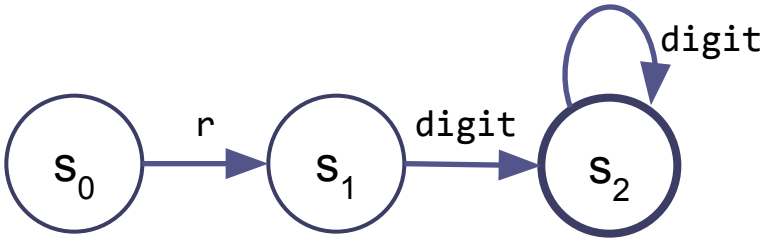
```
r digit digit*
```



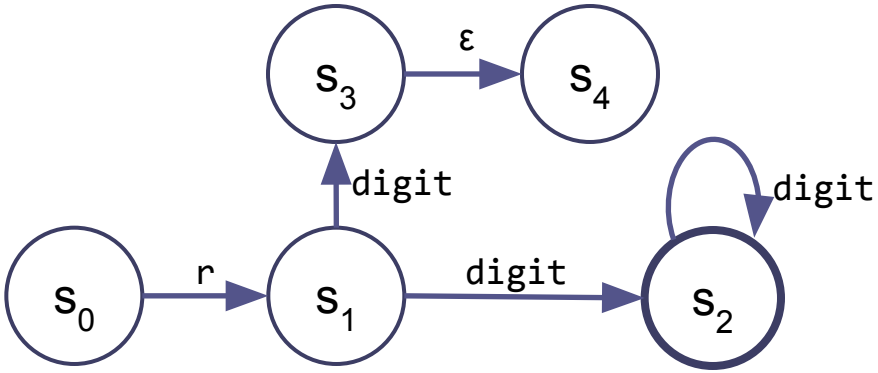
- $Q = \{s_0, s_1, s_2\}$
- $\Sigma = \{r, 0,1,2,3,4,5,6,7,8,9\}$
- $\delta = \{(\langle s_0, r\rangle \to s_1), (\langle s_1, digit\rangle \to s_2), (\langle s_2, digit\rangle \to s_2)\}$
- $q_0 = s_0$
- $F = \{s_2\}$

Детерминированные и недетерминированные КА

Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором нет пустых и неоднозначных переходов между состояниями.



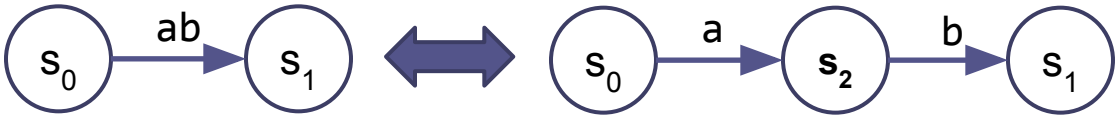
Недетерминированным конечным автоматом (ДКА) называется такой автомат, в котором присутствуют пустые или неоднозначные переходы между состояниями.



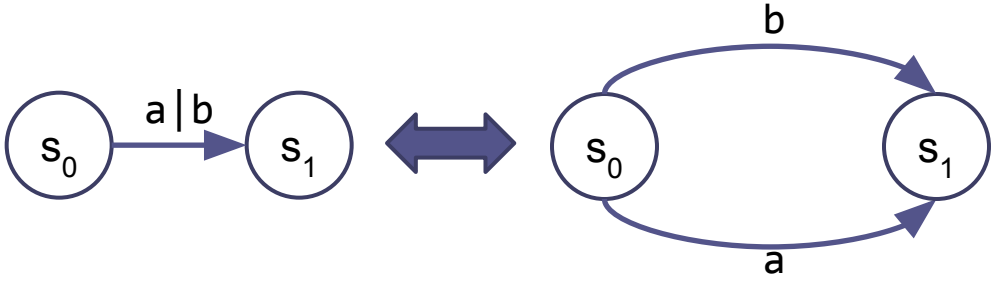
Правила преобразования регулярного выражения в НКА

a, b – регулярные выражения

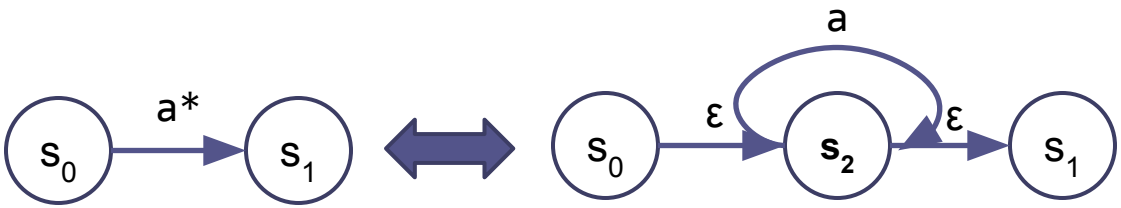
Конкатенация - ab



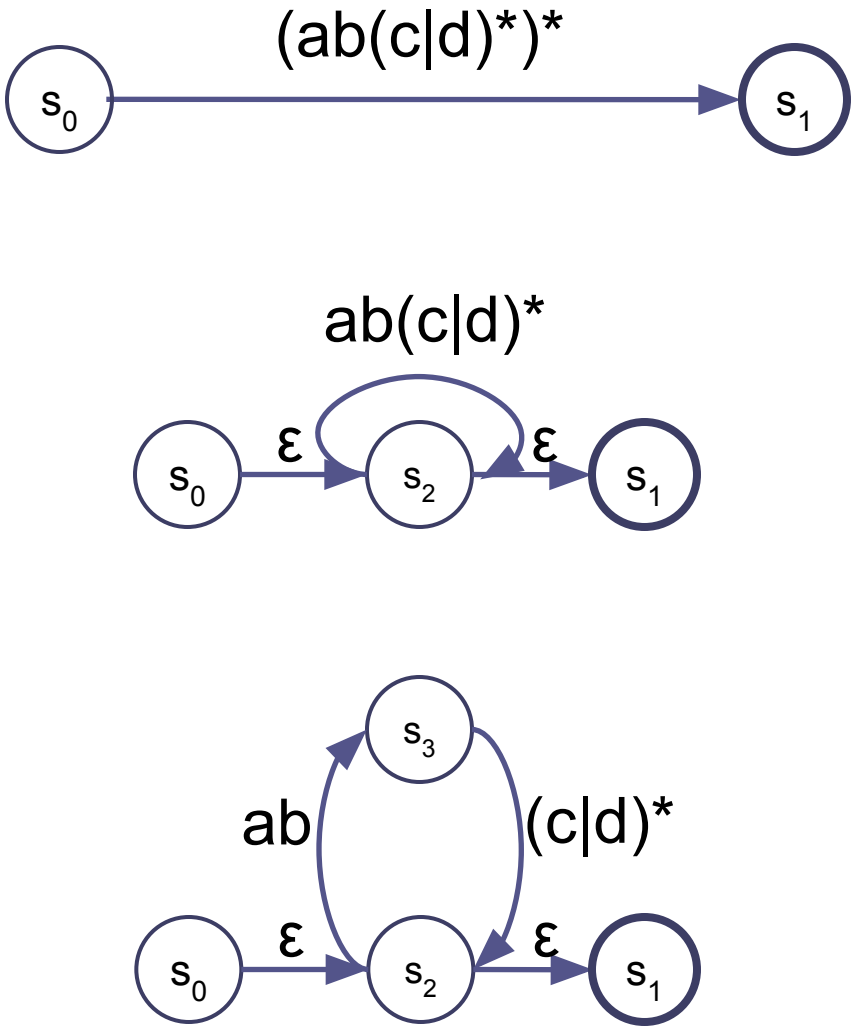
Альтерация – a | b



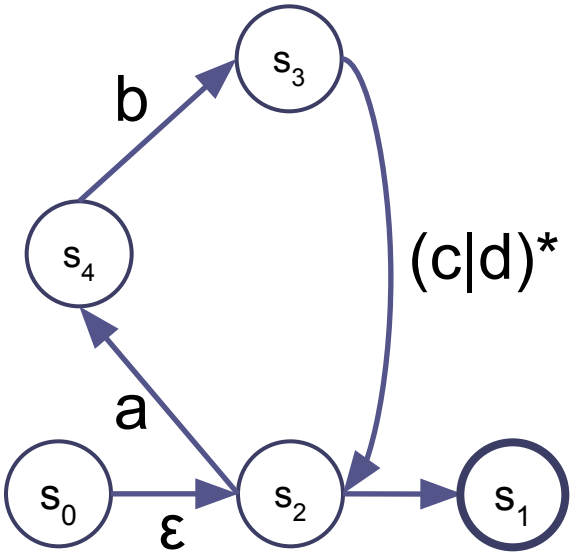
Замыкание Клини – a*

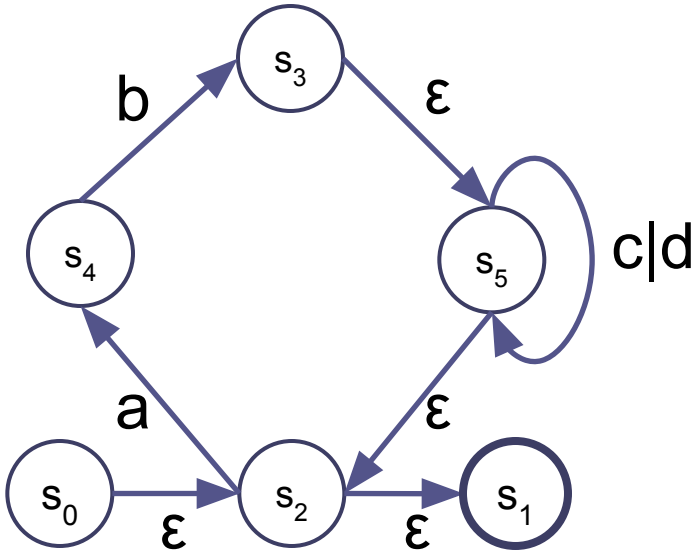


Пример построения НКА



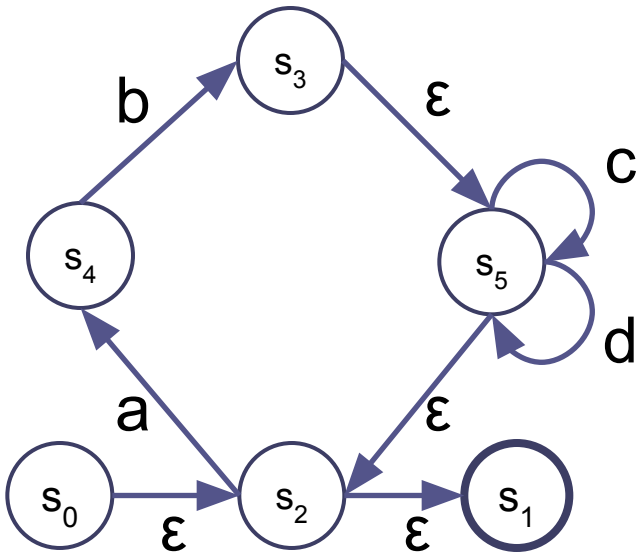
Пример построения НКА





Пример построения НКА

(ab(c|d)*)*



Σ = {a,b,c,d}

Q = {s0, s1, s2, s3, s4, s5}

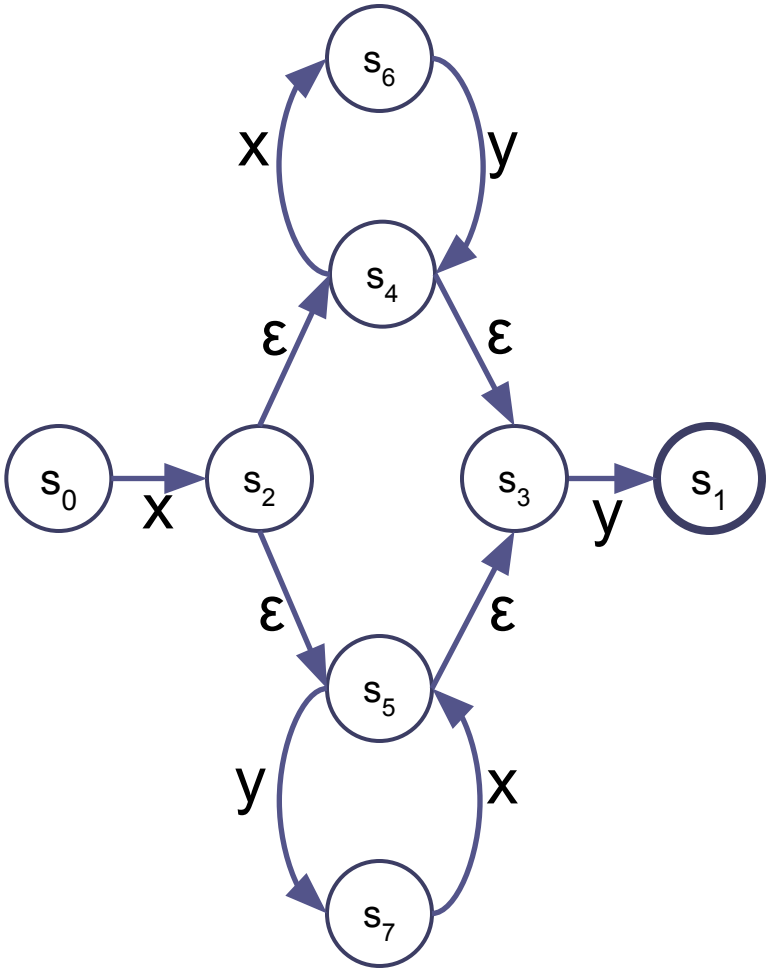
q0 = s0

F = {s1}

δ				
	a	**b**	**c**	**d**
S0	S4	∅	∅	∅
S1*	∅	∅	∅	∅
S2	S4	∅	∅	∅
S3	S4	∅	S5	S5
S4	∅	S3	∅	∅
S5	S4	∅	S5	S5

Пример построения НКА - Упражнение

x((xy)*|(yx)*)y



Σ = {x,y}

Q = {s0, s1, s2, s3, s4, s5, s6, s7}

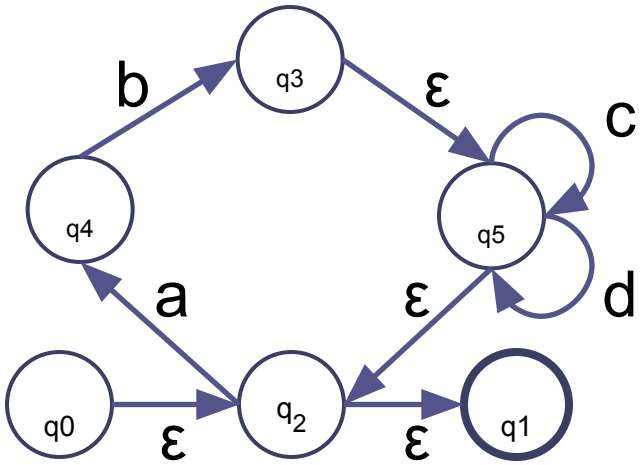
q0 = s0

F = {s1}

δ		
	x	**y**
S0	S2	∅
S1*	∅	∅
S2	S6	{S7,S1}
S3	∅	S1
S4	S6	S1
S5	∅	{S7,S1}
S6	∅	S4
S7	S5	∅

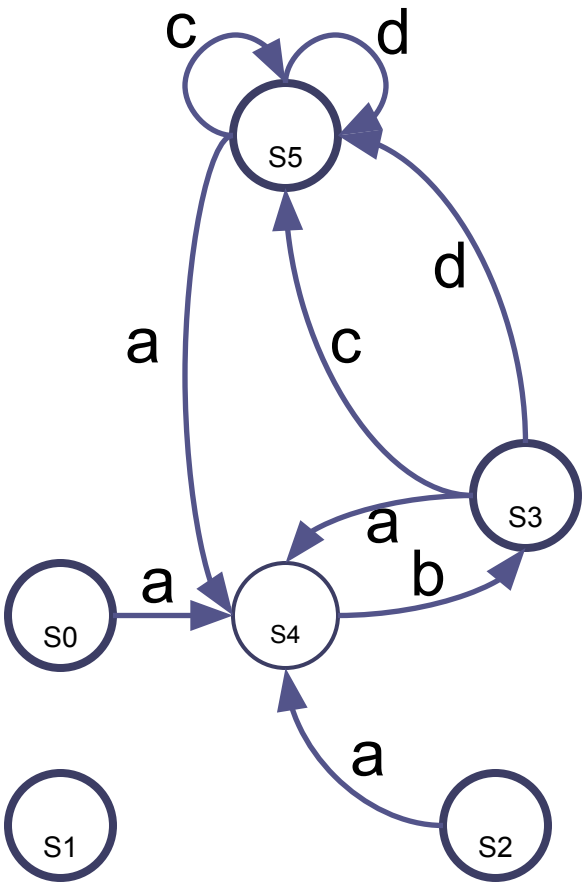
Построение ДКА на основе НКА

(ab(c|d)*)*



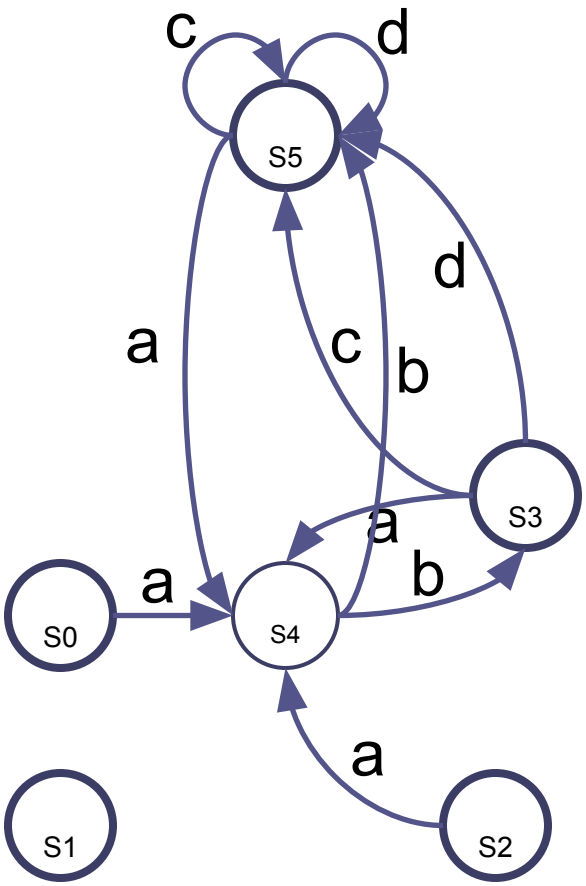
δ				
	a	**b**	**c**	**d**
S0* ≡ {q0} = {q0, q2, q1}	{q4}	∅	∅	∅
S1* ≡ {q1} = {q1}	∅	∅	∅	∅
S2* ≡ {q2} = {q2, q1}	{q4}	∅	∅	∅
S3* ≡ {q3} = {q3, q5, q2, q1}	{q4}	∅	{q5}	{q5}
S4 ≡ {q4} = {q4}	∅	{q3}	∅	∅
S5* ≡ {q5} = {q5, q2, q1}	{q4}	∅	{q5}	{q5}

Построение ДКА на основе НКА (избавление от ε-переходов)



δ				
	a	**b**	**c**	**d**
S0* ≡ {q0} = {q0, q2, q1}	{q4}	∅	∅	∅
S1* ≡ {q1} = {q1}	∅	∅	∅	∅
S2* ≡ {q2} = {q2, q1}	{q4}	∅	∅	∅
S3* ≡ {q3} = {q3, q5, q2, q1}	{q4}	∅	{q5}	{q5}
S4 ≡ {q4} = {q4}	∅	{q3}	∅	∅
S5* ≡ {q5} = {q5, q2, q1}	{q4}	∅	{q5}	{q5}

Построение минимального ДКА



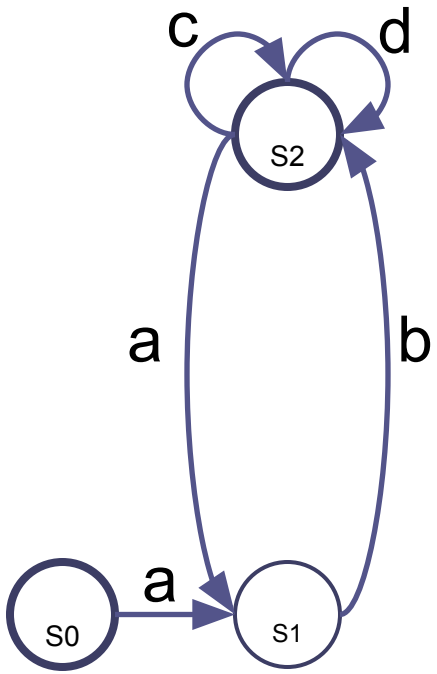
Удаление вершин, в которые мы не можем попасть

Объединение вершин, ведущих в одинаковые состояния и имеющих одинаковый статус (конечные или нет)

δ				
	a	**b**	**c**	**d**
So* ≡ {q0} = {q0, q2, q1}	{q4}	∅	∅	∅
S3* ≡ {q3} = {q3, q5, q2, q1}	{q4}	∅	{q5}	{q5}
S4 ≡ {q4} = {q4}	∅	{q3}	∅	∅
S5* ≡ {q5} = {q5, q2, q1}	{q4}	∅	{q5}	{q5}

Минимальный ДКА

(ab(c|d)*)*



$\Sigma = \{a,b,c,d\}$

$Q = \{s0, s1, s2\}$

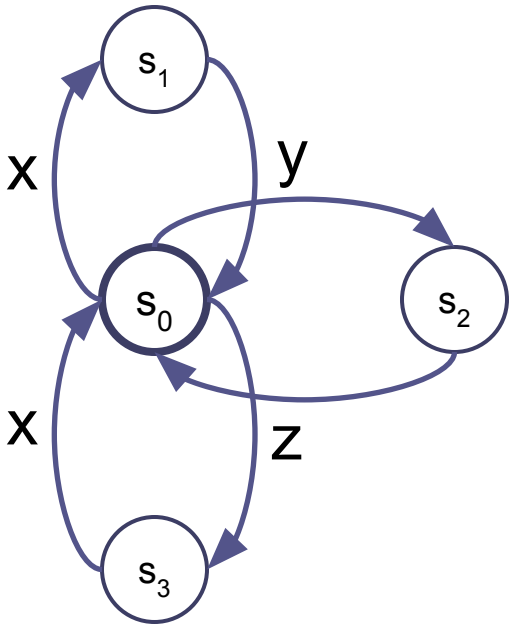
$q0 = s0$

$F = \{s0, s2\}$

δ				
	a	**b**	**c**	**d**
S0*	S1	∅	∅	∅
S1	∅	S2	∅	∅
S2*	S1	∅	S2	S2

Пример построения минимального ДКА - Упражнение

(xy | yz | zx)*



Σ = {x,y,z}

Q = {s0, s1, s2, s3}

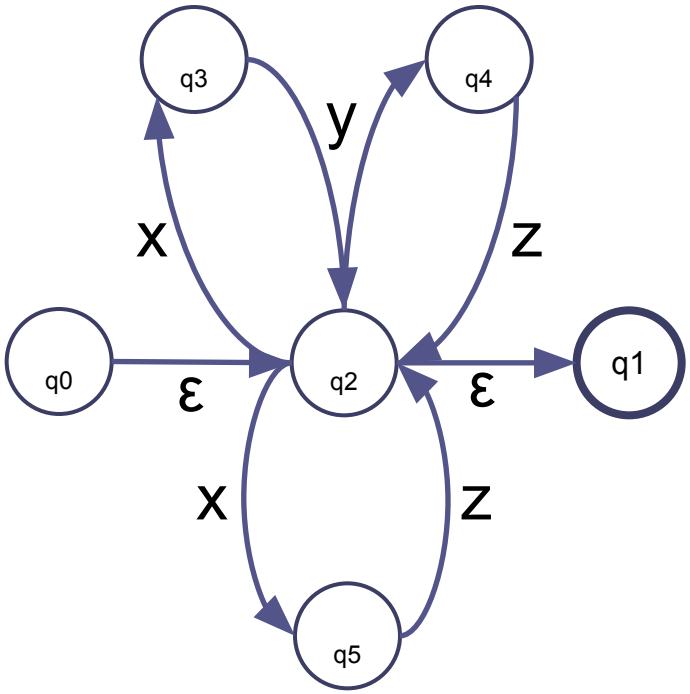
q0 = s0

F = {s0}

δ			
	x	**y**	**z**
So*	S1	S2	S3
S1	∅	So	∅
S2	∅	∅	So
S3	So	∅	∅

Построения ДКА на основе НКА

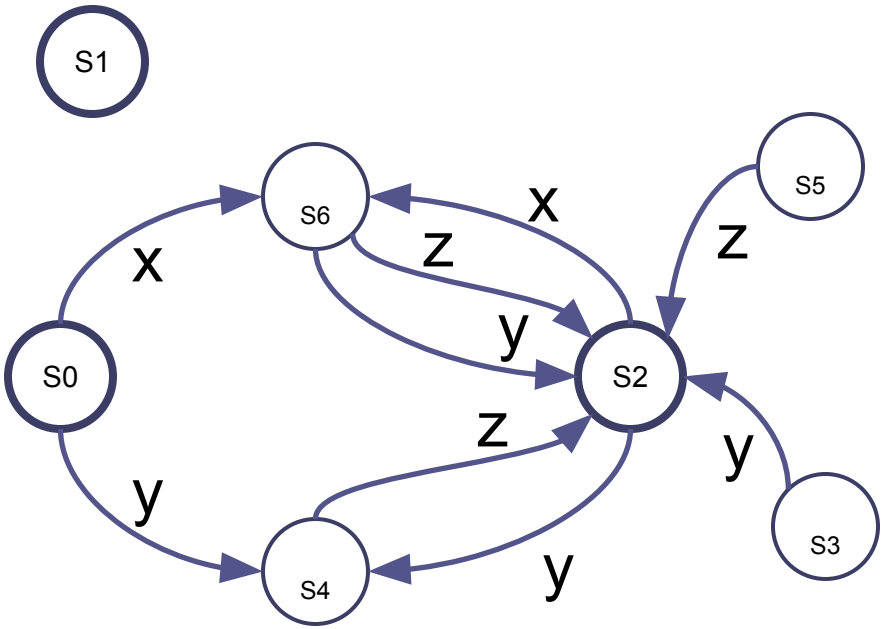
(xy | yz | xz)*



δ			
	x	**y**	**z**
S0* ≡ {q0} = {q0, q2, q1}	{q3, q5}	{q4}	∅
S1* ≡ {q1} = {q1}	∅	∅	∅
S2* ≡ {q2} = {q2, q1}	{q3, q5}	{q4}	∅
S3 ≡ {q3} = {q3}	∅	{q2}	∅
S4 ≡ {q4} = {q4}	∅	∅	{q2}
S5 ≡ {q5} = {q5}	∅	∅	{q2}

Построения ДКА на основе НКА

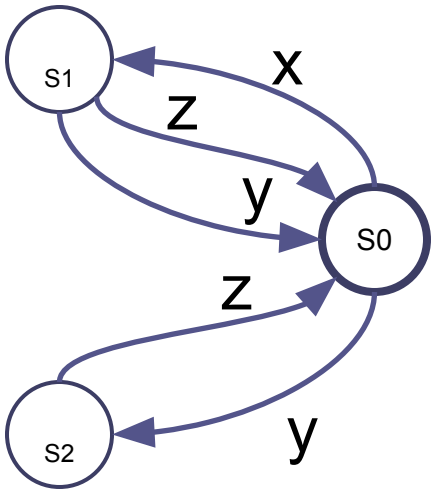
(xy | yz | xz)*



δ			
	x	**y**	**z**
S0* ≡ {q0} = {q0, q2, q1}	{q3, q5}	{q4}	∅
S1* ≡ {q1} = {q1}	∅	∅	∅
S2* ≡ {q2} = {q2, q1}	{q3, q5}	{q4}	∅
S3 ≡ {q3} = {q3}	∅	{q2}	∅
S4 ≡ {q4} = {q4}	∅	∅	{q2}
S5 ≡ {q5} = {q5}	∅	∅	{q2}
S6 ≡ {q3, q5} = {q3, q5}	∅	{q2}	{q2}

Построения минимального ДКА

(xy | yz | xz)*



Σ = {x,y,z}

Q = {s0, s1, s2}

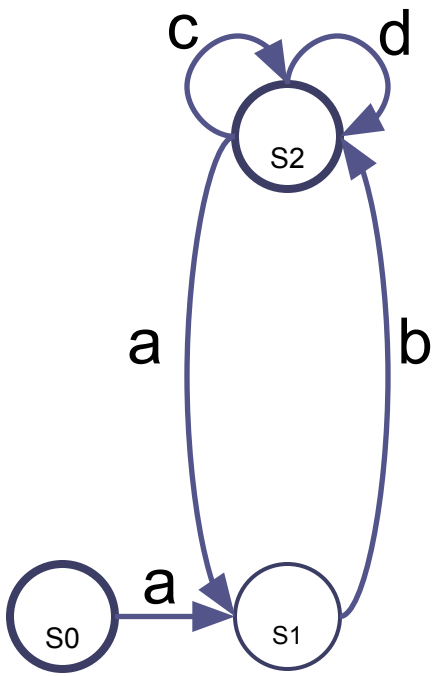
q0 = s0

F = {s0}

δ			
	x	**y**	**z**
So*	S1	S2	∅
S1	∅	So	So
S2	∅	∅	So

Проверка строк на соответствие РВ по ДКА

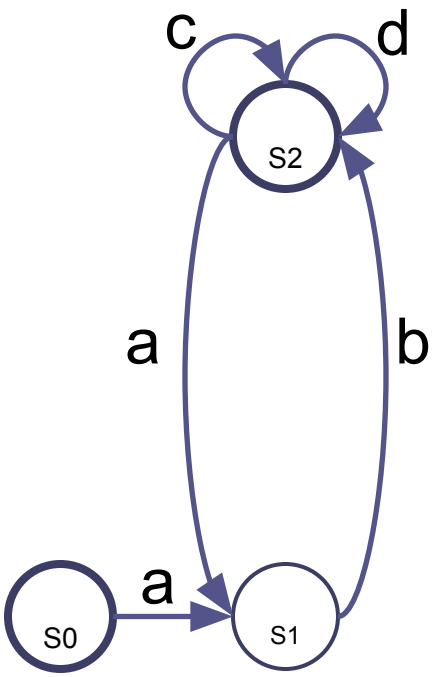
(ab(c|d)*)*



abcabc	Соответствует
abca	Не соответствует
Пустая строка	Соответствует
ab	Соответствует
abx	Не соответствует

Программирование ДКА

(ab(c|d)*)*




```
enum T {S0 = 0, S1 = 1, S2 = 2, ERROR = -1}; // Множество всех состояний
int FStates[] = {S0, S2}; // Конечные состояния
int CurState = 0; // Начальное состояние
char CurCh; // Текущий символ проверяемой строки

// Функция для проверки конечности состояния
bool IsFinish(int State) {
    for (int i : FStates) if (State == i) return true;
    return false;
}

// Функция переходов
int Delta(int State, char Ch) {
    switch (State) {
    case S0:
        if (Ch == 'a') return S1;
        break;
    case S1:
        if (Ch == 'b') return S2;
        break;
    case S2:
        if (Ch == 'c' || Ch == 'd') return S2;
        if (Ch == 'a') return S1;
        break;
    default: break;
    }
    return ERROR;
}
```

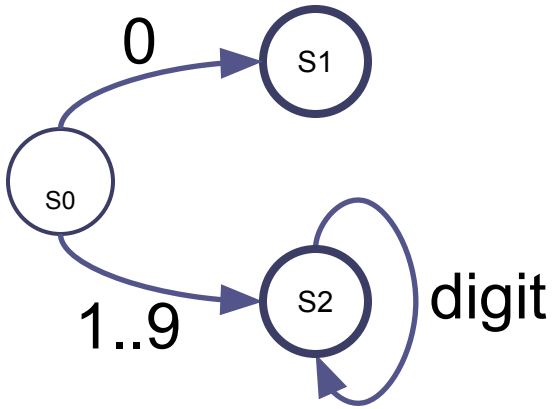
Программирование ДКА (основная функция)

```
int main() {
    ifstream fin("input.txt");
    ofstream fout("output.txt");
    do {
        CurCh = fin.get();
        if (CurCh == '\n') continue;
        if (fin.eof()) break;
        CurState = Delta(CurState, CurCh);
        if (CurState == ERROR) break;
    } while (!fin.eof());

    if (IsFinish(CurState)) fout << "yes" << endl;
    else fout << "no" << endl;


    return 0;
}
```

Минимальные ДКА для основных лексем
Целое неотрицательное число

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

intcon ::= digit | (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) digit*



$\Sigma = \{0..9\}$

$Q = \{s0, s1, s2\}$

$q0 = s0$

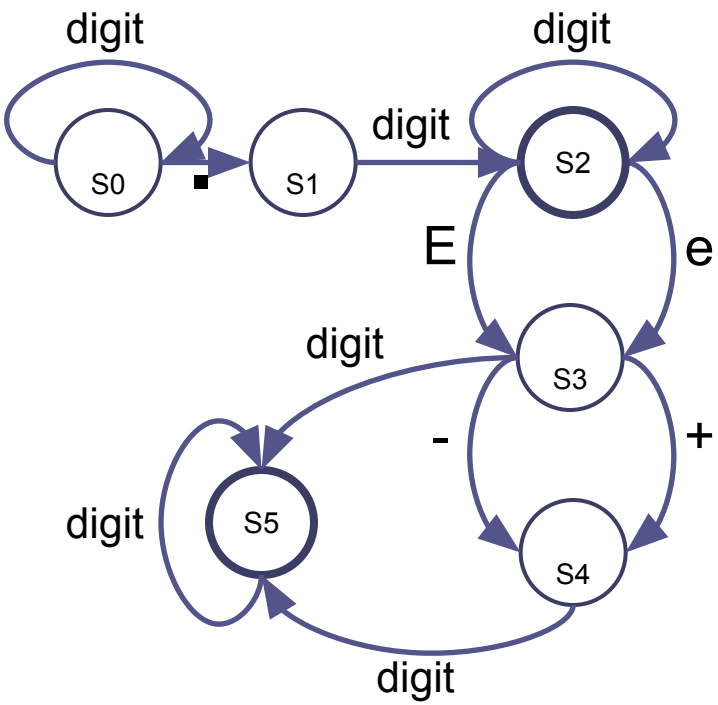
$F = \{s1, s2\}$

δ		
	0	**1..9**
S0	S1	S2
S1*	∅	∅
S2*	S2	S2

Минимальные ДКА для основных лексем
Вещественное положительное число

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

floatcon ::= `digit* . digit digit* (((E|e)(+|-|ε) digit digit*)|ε)`



Σ = {., E, e, +, -, 0..9}

Q = {s0, s1, s2, s3, s4, s5}

q0 = s0

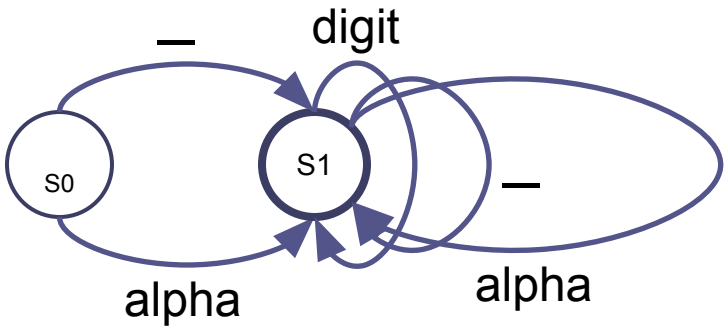
F = {s2, s5}

δ						
	.	E	e	+	-	0..9
S0	S1	∅	∅	∅	∅	S0
S1	∅	∅	∅	∅	∅	S2
S2*	∅	S3	S3	∅	∅	S2
S3	∅	∅	∅	S4	S4	S5
S4	∅	∅	∅	∅	∅	S5
S5*	∅	∅	∅	∅	∅	S5

Минимальные ДКА для основных лексем
Идентификатор

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
alpha ::= a | b | c | d | … | z | A | B | C | D | … | Z

ident ::= (alpha | _) (alpha | digit | _)*



Σ = {0..9, a..zA..z, _}

Q = {s0, s1}

q0 = s0

F = {s1}

δ			
	_	0..9	alpha
So	S1	∅	S1
S1*	S1	S1	S1

Детерминированный синтаксис

1. Если две лексемы описываются регулярными выражениями re_1 и re_2, то
   $\text{First}(re_1) \cap \text{First}(re_2) = \varnothing$.
   Здесь First(*re*) – это множество символов, содержащее все символы, с которых может начинаться цепочка, соответствующая регулярному выражению *re*.

2. Если лексема начинается с регулярного выражения вида (*re**), то
   $\text{First}(re) \cap \text{Follow}(re) = \varnothing$.
   Здесь Follow(*re*) – это множество символов, которые могут следовать за регулярным выражением (*re**).

```
ident ::= (alpha | _ ) ( alpha | digit | _ )*
intcon ::= digit digit*
symbol ::= + | – | * | / | ( | ) | :=
```

Реализация анализатора по набору ДКА для детерминированного синтаксиса

```
ident ::= (alpha | _ ) ( alpha | digit | _ )*
intcon ::= digit digit*
symbol ::= + | – | * | / | ( | ) | :=
```

Посимвольно читаем входной поток

Проверяем текущий символ на соответствие первому символу одного из регулярных выражений

Распознаем лексему с помощью соответствующего регулярному выражению ДКА

Если ни один ДКА не позволил выявить тип токена, сигнализируем об ошибке

Реализация анализатора по набору ДКА для недетерминированного синтаксиса

```
ident ::= (alpha | _ ) ( alpha | digit | _ )*
floatcon ::= digit* . digit digit* (((E|e)(+|-|ε) digit digit*)|ε)
intcon ::= digit digit*
symbol ::= + | – | * | / | ( | ) | :=
```

Посимвольно читаем входной поток

Проверяем текущий символ на соответствие первому символу одного из регулярных выражений

Если текущий символ соответствует началу нескольких регулярных выражений, проверяем их с помощью соответствующих ДКА в порядке приоритета

Если ни один ДКА не позволил выявить тип токена, сигнализируем об ошибке

Пример лексического разбора с помощью ДКА

```
ident ::= (alpha | _ ) ( alpha | digit | _ )*
intcon ::= digit digit*
symbol ::= + | – | * | / | ( | ) | :=
```

```
/* Идентификатор */
if (isalpha(ch) || ch == '_') {
    int CurState = S0, PrevState = SERR;
    string CurToken = "";
    CurState = DeltaIdent(ch, CurState);
    while (CurState != SERR) {
        if (CurToken.length() < MAXIDENT) CurToken += ch;
        PrevState = CurState;
        ch = getchar();
        CurState = DeltaIdent(ch, CurState);
    }

    if (IsFinish(PrevState, FStatesIdent)) {
        sym = T_IDENT;
        ident = CurToken;
    }
    else {
        sym = T_UNDEF;
    }
}
```

```
/* Целочисленная константа */
else if (isdigit(ch)) {
    int CurState = S0, PrevState = SERR;
    string CurToken = "";
    CurState = DeltaIntcon(ch, CurState);
    while (CurState != SERR) {
        CurToken += ch;
        PrevState = CurState;
        ch = getchar();
        CurState = DeltaIntcon(ch, CurState);
    }
    if (IsFinish(PrevState, FStatesIntcon)) {
        sym = T_INTCON;
        num = stoi(CurToken);
    }
    else {
        sym = T_UNDEF;
    }
}
```

Реализация анализатора непосредственно по РВ

re	*P(re)*

'x'	`if (sym == 'x')`
	`  sym = getchar ();`
	`else`
	`  error ();`

(exp)	`P(exp)`
exp*	`while (sym IN FIRST(exp))` `  P(exp);`
exp1 exp2 exp3	`P(exp1); P(exp2); P(exp3);`

exp1 | exp2 | exp3	`if (sym IN FIRST(exp1))`
	`  P(exp1);`
	`else if (sym IN FIRST(exp2))`
	`  P(exp2);`
	`else if (sym IN FIRST(exp3))`
	`  P(exp3);`

Реализация анализатора непосредственно по РВ. Пример

a(b*c) | bc*

```
if (sym IN FIRST(‘a(b*c)’))
    P(‘a(b*c)’);
else if (sym IN FIRST(‘bc*’))
    P(‘bc*’);
```

```
if (sym == ‘a’)
    P(‘a’); P(‘b*’); P(‘c’);
else if (sym == ‘b’)
    P(‘b’); P(‘c*’);
```

```
if (sym == ‘a’) {
    if (sym == ‘a’) sym = getchar();
    else error();
    while (sym IN FIRST(‘b’))
        P(‘b’);
    if (sym == ‘c’) sym = getchar();
    else error();
}
else if (sym == ‘b’) {
    if (sym == ‘b’) sym = getchar();
    else error();
    while (sym IN FIRST(‘c’))
        P(‘c’);
}
```

```
if (sym == ‘a’) {
    if (sym == ‘a’) sym = getchar();
    else error();
    while (sym == ‘b’)
        if (sym ==‘b’) sym = getchar();
        else error();
    if (sym == ‘c’) sym = getchar();
    else error();
}
else if (sym == ‘b’) {
    if (sym == ‘b’) sym = getchar();
    else error();
    while (sym == ‘c’)
        if (sym ==‘c’) sym = getchar();
        else error();
}
```

Реализация анализатора непосредственно по РВ
Идентификатор

ident ::= (alpha | _) (alpha | digit | _)*

```
P( alpha | _ );
P( (alpha | digit | _)* );
```

```
if (sym IN FIRST( alpha )) P(alpha);
else if (sym IN FIRST( _ )) P('_');
while (sym IN FIRST( alpha | digit | _ ))
        P( alpha | digit | _ );
```

```
if (isalpha(sym))
    if (sym IN FIRST( alpha ))
       sym = getchar();
    else error();
else if (sym IN FIRST( _ ))
    if (sym IN FIRST( _ ))
       sym = getchar();
    else error();
while (isalpha(sym) ||
       isdigit(sym) ||
       sym == '_')
    if (sym IN FIRST( alpha ))
       P( alpha );
    else if (sym IN FIRST( digit ))
       P( digit );
    else if (sym IN FIRST( _ ))
       P( _ );
```

```
if (isalpha(sym))
    if (isalpha(sym)) sym = getchar();
    else error();
else if (sym == '_')
    if (sym == '_') sym = getchar();
    else error();

while (isalpha(sym) || isdigit(sym) || sym == '_')
    if (isalpha(sym))
       if (isalpha(sym)) sym = getchar();
       else error();
    else if (isdigit(sym))
       if (isdigit(sym)) sym = getchar();
       else error();
    else if (sym == '_')
       if (sym == '_') sym = getchar();
       else error();
```

Реализация анализатора непосредственно по РВ
Вещественное положительное число

floatcon ::= digit* . digit digit* ((E|e)(+|-|ε) digit digit*)|ε

```
while (isdigit(sym))
    if (isdigit(sym)) sym = getchar();
    else error();
if (sym == '.') sym = getchar();
else error();
if (isdigit(sym)) sym = getchar();
else error();
while (isdigit(sym))
    if (isdigit(sym)) sym = getchar();
    else error();
if (sym == 'E' || sym == 'e') {
    if (sym == 'E' || sym == 'e') sym = getchar();
    else error();
    if (sym == '+' || sym == '-') sym = getchar();
    if (isdigit(sym)) sym = getchar();
    else error();
    while (isdigit(sym))
        if (isdigit(sym)) sym = getchar();
        else error();
}
```

```
while (isdigit(sym)) sym = getchar();

if (sym == '.') sym = getchar();
else error();

if (isdigit(sym)) sym = getchar();
else error();

while (isdigit(sym)) sym = getchar();

if (sym == 'E' || sym == 'e') {
    sym = getchar();

    if (sym == '+' || sym == '-')
        sym = getchar();

    if (isdigit(sym)) sym = getchar();
    else error();

    while (isdigit(sym)) sym = getchar();
}
```

Пример лексического разбора с помощью РВ

```
ident ::= (alpha | _ ) ( alpha | digit | _ )*
intcon ::= digit digit*
symbol ::= + | – | * | / | ( | ) | :=
```

```
/* Идентификатор */
if (isalpha(ch) || ch == '_') {
    string CurToken = "";
    do {
        if (CurToken.length() < MAXIDENT)
            CurToken += ch;
        ch = getchar();
    } while (
        isalpha(ch) ||
        isdigit(ch) ||
        ch == '_');
    sym = T_IDENT;
    ident = CurToken;
}
```

```
/* Целочисленная константа */
else if (isdigit(ch)) {
    string CurToken = "";
    do {
        CurToken += ch;
        ch = getchar();
    } while (isdigit(ch));
    sym = T_INTCON;
    num = stoi(CurToken);
}
```

Примеры работы лексического анализатора

```
a := b + 234 /
    (5 * y - 10-100*
        (abc + def))
```

```
T_IDENT, a
T_ASSGN, -
T_IDENT, b
T_ADD, -
T_INTCON, 234
T_DIV, -
T_LPAREN, -
T_INTCON, 5
T_MUL, -
T_IDENT, y
T_SUB, -
T_INTCON, 10
T_SUB, -
T_INTCON, 100
T_MUL, -
T_LPAREN, -
T_IDENT, abc
T_ADD, -
T_IDENT, def
T_RPAREN, -
T_RPAREN, -
```

```
abc...
```

```
T_IDENT, abc
T_UNDEF, -
T_UNDEF, -
T_UNDEF, -
```

Генератор лексических анализаторов LEX/FLEX

http://flex.sourceforge.net/