



Автоматы и формальные языки

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского Политехнического университета

karpov@dcn.infos.ru



Структура курса

- Конечные автоматы-распознаватели – 4 л
 - Лекция 1. Формальные языки. Примеры языков. Грамматики. КА
 - Лекция 2. Теория конечных автоматов-распознавателей
 - Лекция 3. Трансляция автоматных языков
 - Лекция 4. Регулярные множества и регулярные выражения
- Порождающие грамматики Хомского – 3 л
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции 2 л

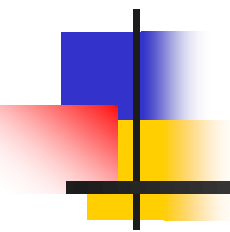


Объявление

Начиная с 5 марта
по средам с 17 часов в к.106 корп. 9
студент 6 курса

Руслан Морозов

проводит консультации по решению задач и
выполнению курсовых работ



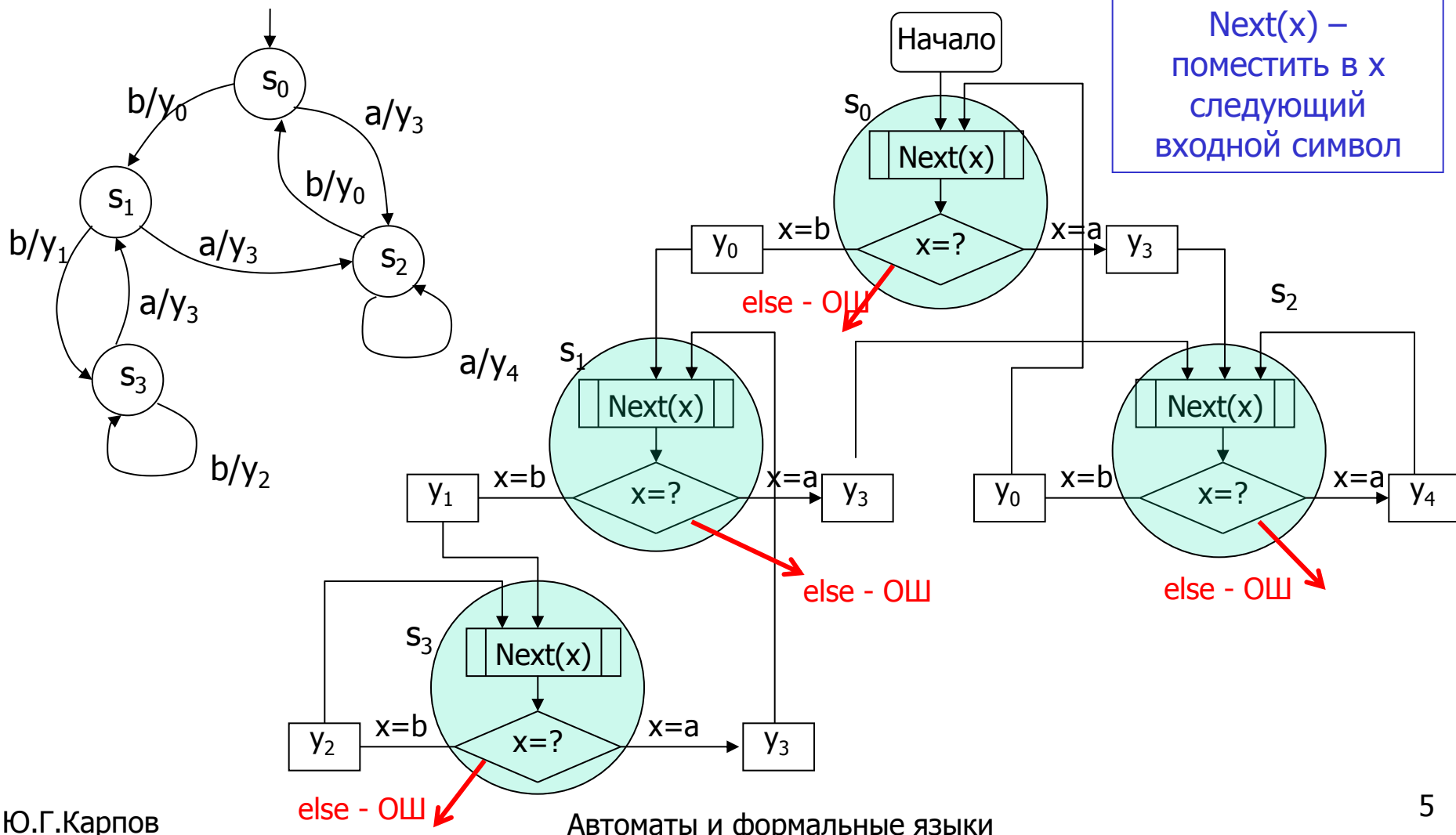
Реализация конечных автоматов- распознавателей

Любой детерминированный конечный автомат может быть реализован программно или аппаратно множеством разных способов.

Методы программной реализации КА являются простой технической проблемой

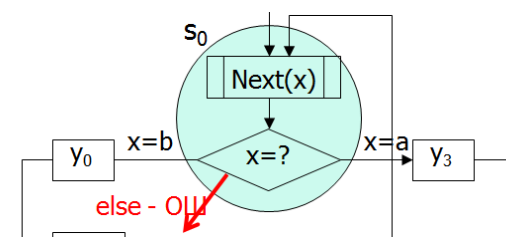
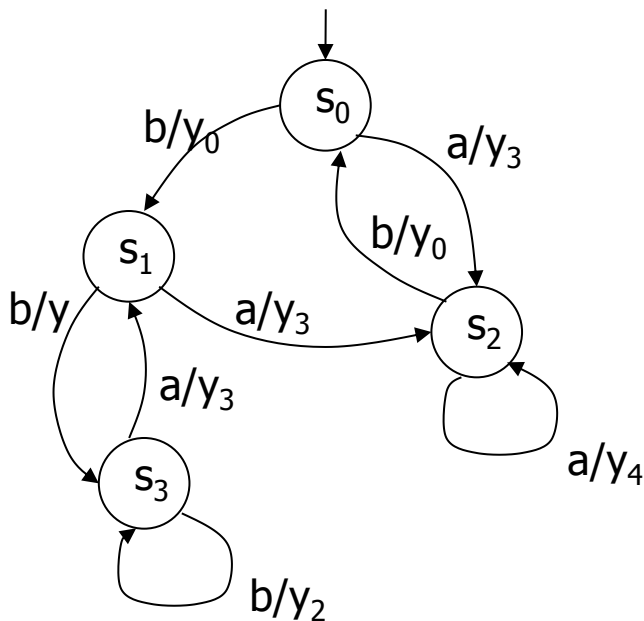
Программная реализация КА вручную: блок-схемы

- В каждом состоянии – ждать входной символ и проверить, какой он
- На каждом переходе: выполнить действие и определить новое состояние (переход)



Программная реализация КА на псевдокоде: if-then

В бесконечном цикле: в каждом очередном состоянии ждать входной символ и для каждого возможного входного символа выполнить действие и установить новое состояние



```

begin
enum States {STATE_0, STATE_1, STATE_2, STATE_3 };
char x;
States S = STATE_0; /* установка нач состояния */
while true do      /* бесконечный цикл */
    Next(x);        /* в x – очередной входной символ */
    if S= STATE_0 then
        if x='a' then y3(); S:= STATE_2; break fi;
        if x='b' then y0(); S:= STATE_1; break fi;
        Err0( )     /* ОШ: в сост s0 вход – не а и не b */
    fi;
    if S= STATE_1 then
        if x='a' then y3(); S:= STATE_2; break fi;
        if x='b' then y1(); S:= STATE_3; break fi;
        Err1( )
    fi;
    ... ..
od;
end
  
```

Реализация динамики работы КА с использованием переключателя

В бесконечном цикле: для текущего состояния:
а) вводим входной сигнал,
б) распознаем его,
в) выполняем действие на переходе,
г) сменяем состояние.

Возможных реализаций КА много, они почти очевидны

```
#define STATE_0    0
...
#define STATE_3    3

main( ) {
int state = STATE_0 ;      /* текущее состояние, вначале  $s_0$  */
char x;                    /* сюда - очередной символ из входного потока */
while( True ) {           /* бесконечный цикл: автомат постоянно ждет входа */
    x = fgetchar (input);  /* читаем следующий входной символ */
    switch (state) {       /* переключатель по текущему состоянию */
        case STATE_0:     /* Если очередное состояние – STATE_0, то: */
            switch(x) {    /* переключатель по очередному входному символу */
                case 'a':  /* если вход a, то выполняем выходное действие y3 */
                    y3( ); /* переходим в следующее состояние STATE_2 */
                    state = STATE_2; /* конец обработки альтернативы */
                    break;
                case 'b':  /* если вход b, то выполняем выходное действие y0 */
                    y0( ); /* переходим в новое состояние STATE_1 */
                    state = STATE_1;
                    break;
                else:      /* если в состоянии STATE_0 вход не a и не b, то ошибка */
                    Err0( );
            }; /* end switch x */
        case STATE_1:
            ...
    }; /* end switch state */
}; /* end while */
} /* end main */
```

switch – технология:

один из очевидных наборов правил ручного перевода графа или таблицы КА в код с оператором switch

Реализация КА с настраиваемой функцией переходов и выходов

- Таблица переходов и выходов **настраивается** для конкретного автомата
- Engine – **ДВИЖОК** – **всегда один и тот же** для всех автоматов:
 - 1) хранит номер текущего состояния,
 - 2) читает на входе очередной символ
 - 3) выполняет переходы и действия в соответствии с ТПВ
- После настройки ТПВ получаем ТРАНСЛЯТОР, и он обрабатывает ВСЕ входные цепочки

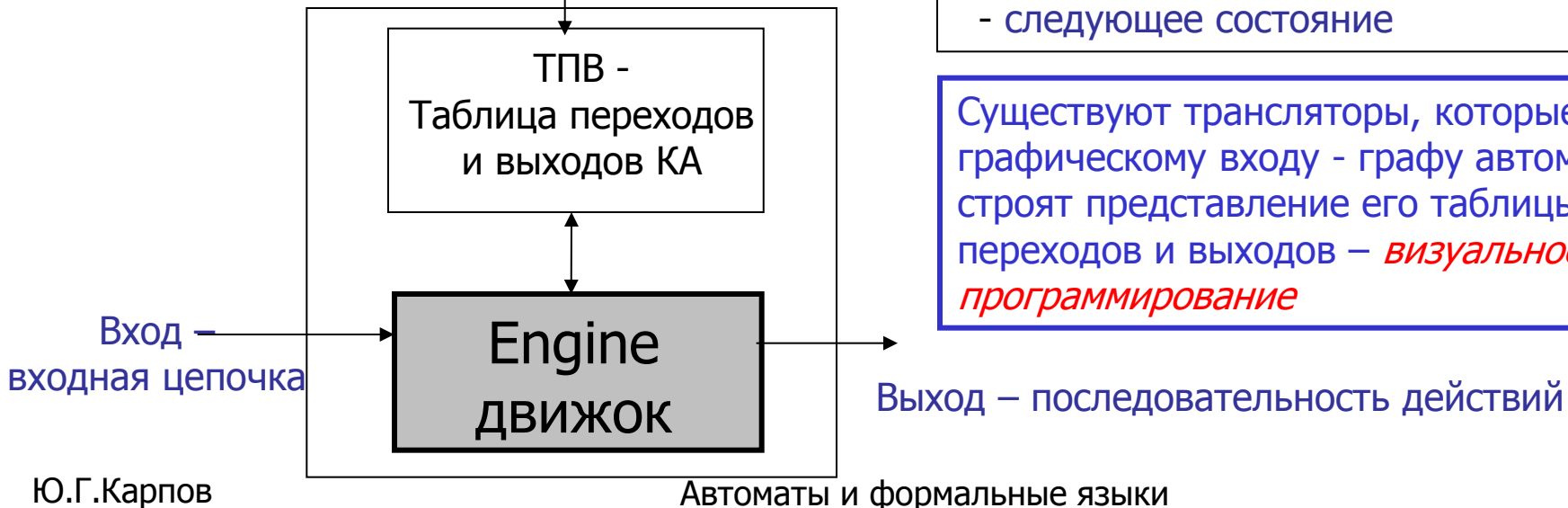
**Настройка автомата –
создание внутреннего
представления таблицы
переходов и выходов**


ввод таблицы
или графа
переходов и
выходов

Обращение к таблице переходов и выходов автомата по текущему состоянию и входу возвращает пару:

- выполняемую на переходе функцию
- следующее состояние

Существуют трансляторы, которые по графическому входу - графу автомата - строят представление его таблицы переходов и выходов – **визуальное программирование**

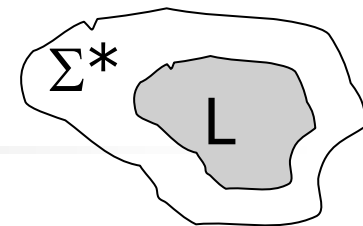




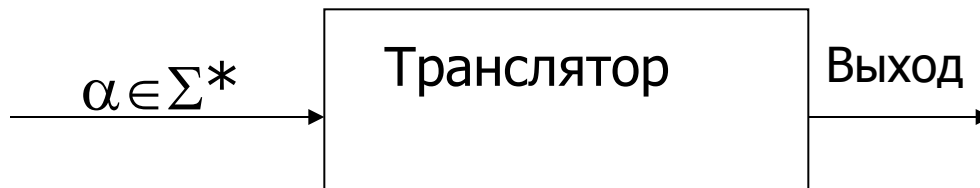
Построение трансляторов автоматных языков

Автоматная спецификация – это не просто словесное описание алгоритма – это формальная выполняемая (executable) спецификация.

“Формальная” означает возможность тестирования, верификации и последующей трансформации введенной спецификации в код



- Распознавание – это только одна из операций над языками. Другой операцией является трансляция



- **Определение. Трансляция - это перевод цепочек языка в некоторый выход. Если цепочка НЕ является цепочкой языка, то выдается ошибка. Если входная цепочка принадлежит языку, то выдается требуемый результат. Этот результат может быть любым:**
 - текст на языке машинных команд;
 - текст на языке ассемблера;
 - непосредственное выполнение нужной последовательности действий;
 - картинка;
 - ...
- **Трансляция формальных языков всегда основывается на распознавании структуры входных цепочек (предложений языка)**



Синтаксис и семантика

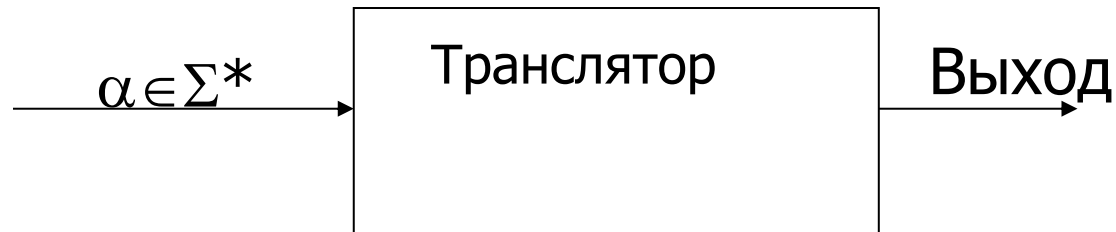
- **Синтаксис** – это правила построения конструкций языка, структуры его предложений (цепочек).
- Синтаксис задается грамматикой (в том или ином виде)
- **Семантика** – это правила понимания, интерпретации предложений и конструкций языка
- Неформально: семантика – правила отображения множества конструкций языка на произвольное множество (которое можно назвать множеством смыслов)
- Семантика в практике трансляции задается действиями, выполняемыми при распознавании конструкций языка

Синтаксис – это ФОРМА,
структура предложений языка

Семантика – это СОДЕРЖАНИЕ,
смысл предложений языка



Трансляция автоматных языков



- Трансляторы **автоматных** языков могут строиться только на основе **детерминированного** конечного автомата-распознавателя добавлением семантических операций – действий на переходах
- Выходом транслятора автоматного языка является обычно последовательность действий по преобразованию входной цепочки символов в некоторые внутренние структуры данных для получения нужного результата



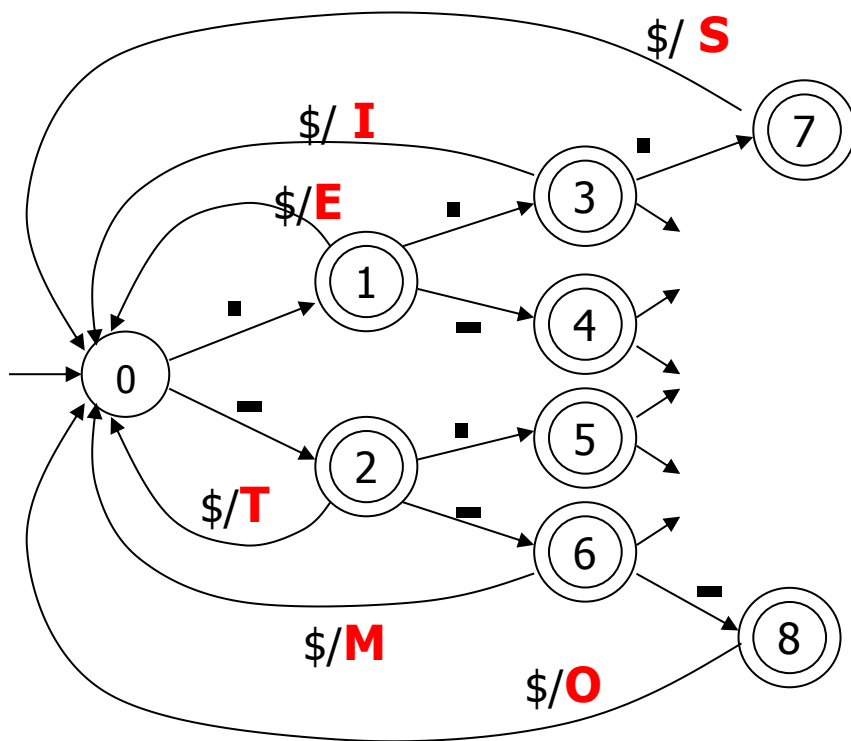
Трансляторы автоматных языков

- Реализация распознавателей автоматных языков проста: по графу переходов детерминированного конечного автомата строится программа либо в виде блок-схемы, либо на каком-нибудь языке, либо в виде перенастраиваемой таблицы переходов и выходов автомата, которую интерпретирует “движок” – engine
- Транслятор строится добавлением семантических действий (операций), которые выполняются по мере распознавания входной строки

Транслятор автоматного языка – это, фактически, реализация конечного автомата с выходом

Транслятор азбуки Морзе

- Добавив выходы к некоторым переходам, можем получить не только распознавание, но и трансляцию
- Выходы назовем семантическими функциями (здесь нет "внутренних" данных)



A . . .
B
C
D . . .
E .
F
G
H
I . .
J
K
L
M
N . . .
O
P
Q
R
S . . .
T . . .

U
V
W
X
Y
Z

1
2
3
4
5
6
7
8
9
0

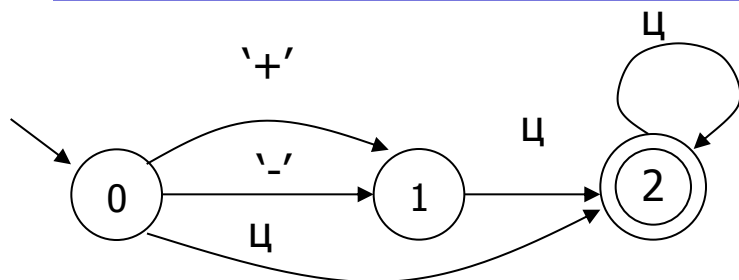
...\$---\$...\$ → Транслятор → SOS

Транслятор языка целых чисел



Распознавание и трансляция языка целых чисел: $\{+23, -23056, 09880, \dots\}$

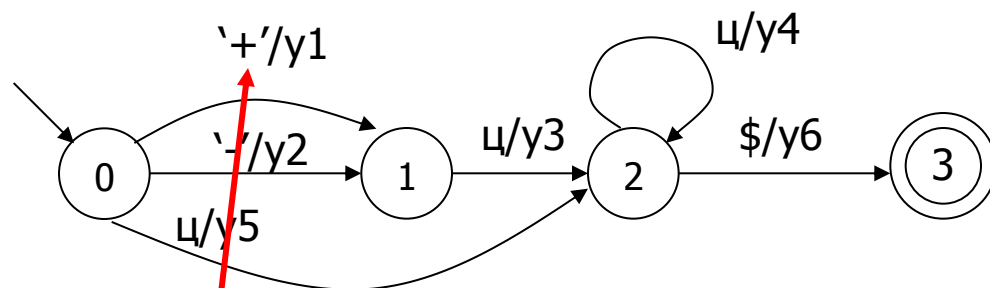
Строим граф переходов
детерминированного
автомата-распознавателя



По графу ДКА можно
построить алгоритм
распознавания
правильности структуры
входной цепочки

$\varphi(\text{ц})$ – по символу входной цифры
выдается двоичное число,
изображаемое этой цифрой

Добавляем действия на переходах



y1: $x:=0; z:=1;$

y4: $x:=\varphi(\text{ц})+10*x;$

y2: $x:=0; z:=-1;$

y5: $x:=\varphi(\text{ц}); z:=1;$

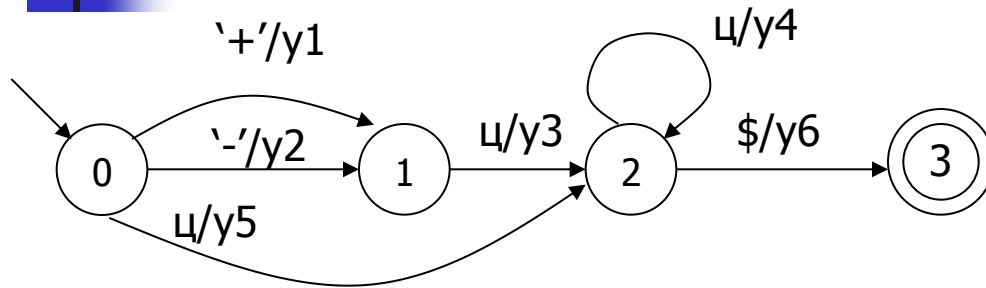
y3: $x:=\varphi(\text{ц});$

y6: $X:=x * z;$

'+' и + : в чем разница?

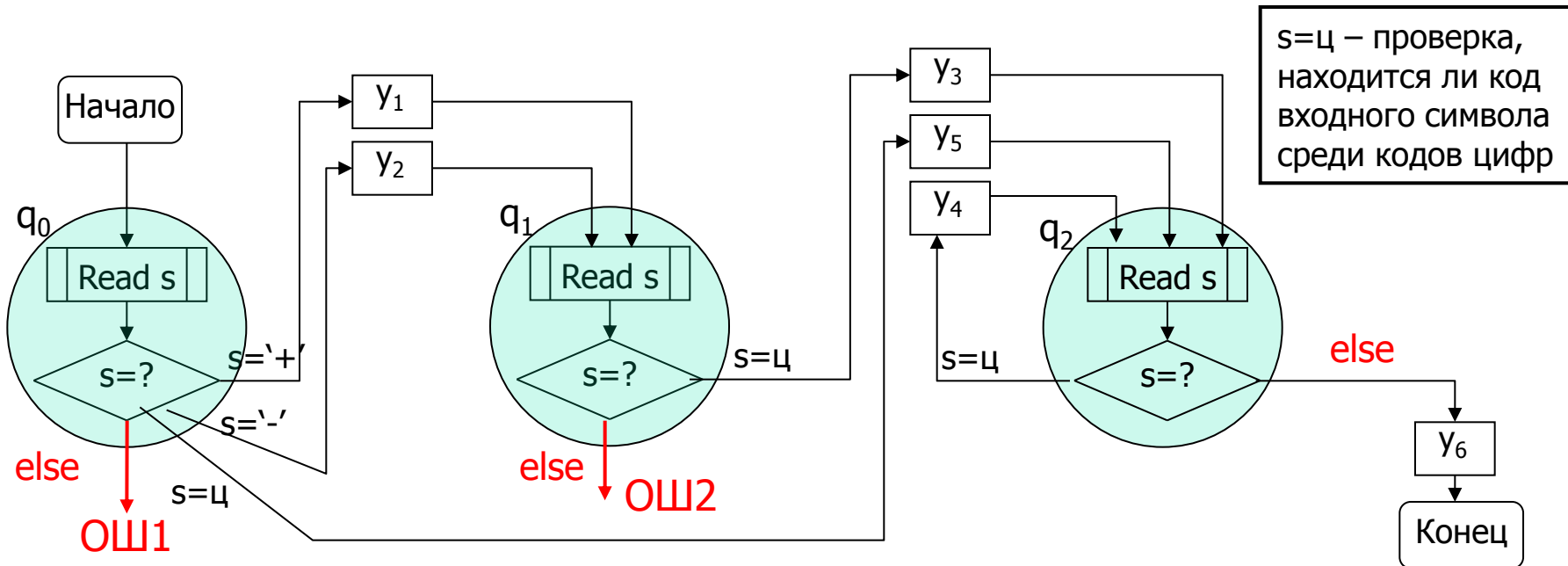
Транслятор – конечный автомат с
выходом (преобразователь)

Выявление ошибок при трансляции автоматных языков



Ош1: число должно начинаться со знака или цифры, а у Вас ... !

Ош2: после знака в целом числе может идти только цифра



По ДЕТЕРМИНИРОВАННОМУ конечному автомату с выходом всегда можно построить программу – транслятор данного автоматного языка

Транслятор языка вещественных чисел

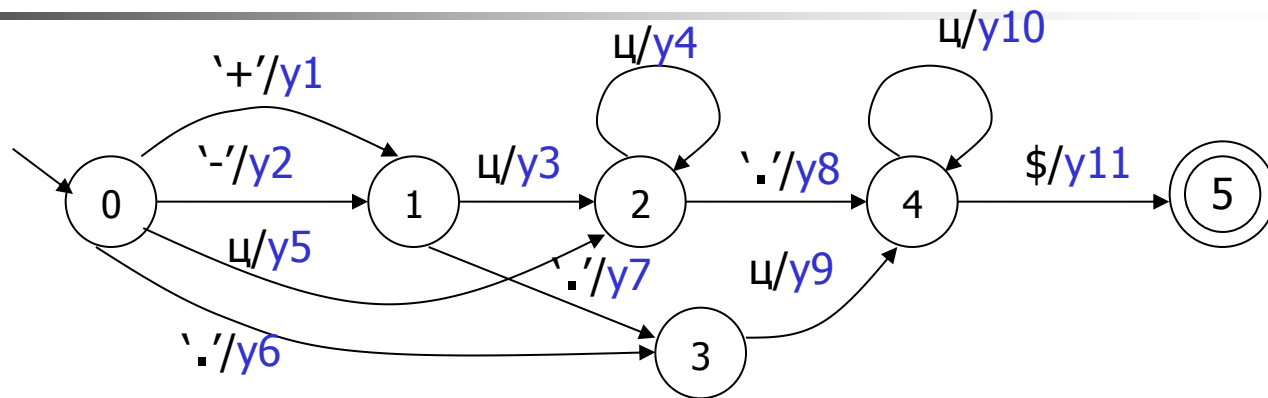
+92.009

.345

-.002

+.0

0.



y1: x:=0; d:=0; z:=1;

y2: x:=0; d:=0; z:=-1;

y3: x:=φ(ц);

y4: x:=φ(ц) + 10*x;

y5: x:=φ(ц); d:=0; z:=1;

y6: x:=0; d:=0; z:=1;

y7: x:=0; d:=0;

y8: k:=0.1;

y9: d:= φ(ц)/10; k:=0.01;

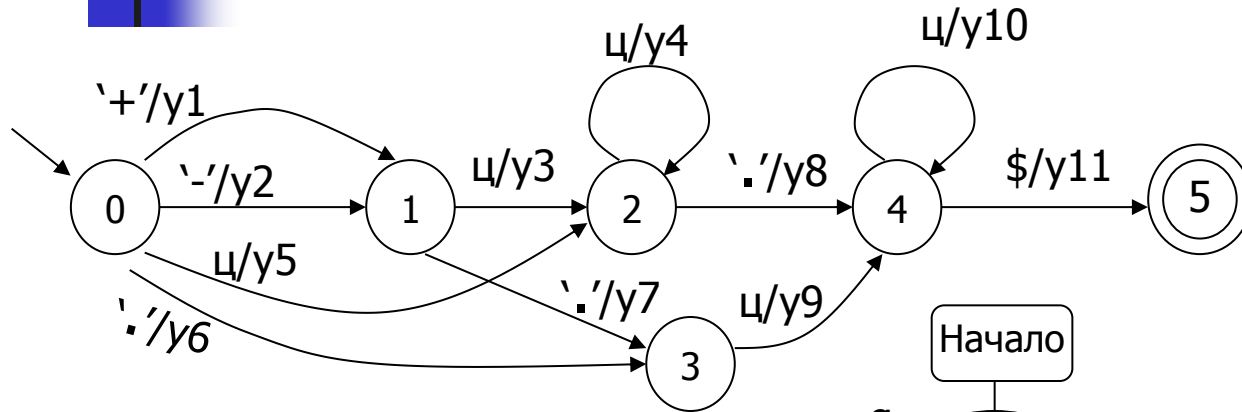
y10: d:=d + φ(ц)*k; k:=k/10

y11: X:= z*(x+d)

φ(ц) – по символу
цифры выдается
двоичное число,
изображаемое этой
цифрой

По цепочке -.002\$ будет выдана последовательность действий (операций) y2 y7 y9 y10 y10 y11. Фактически, входная цепочка сама управляет выполнением последовательности семантических действий при трансляции (text-directed processing – обработка, управляемая самим входным текстом)

Ошибки при трансляции языка вещественных чисел



Ош1: число должно начинаться со знака, точки или цифры. А у Вас ...

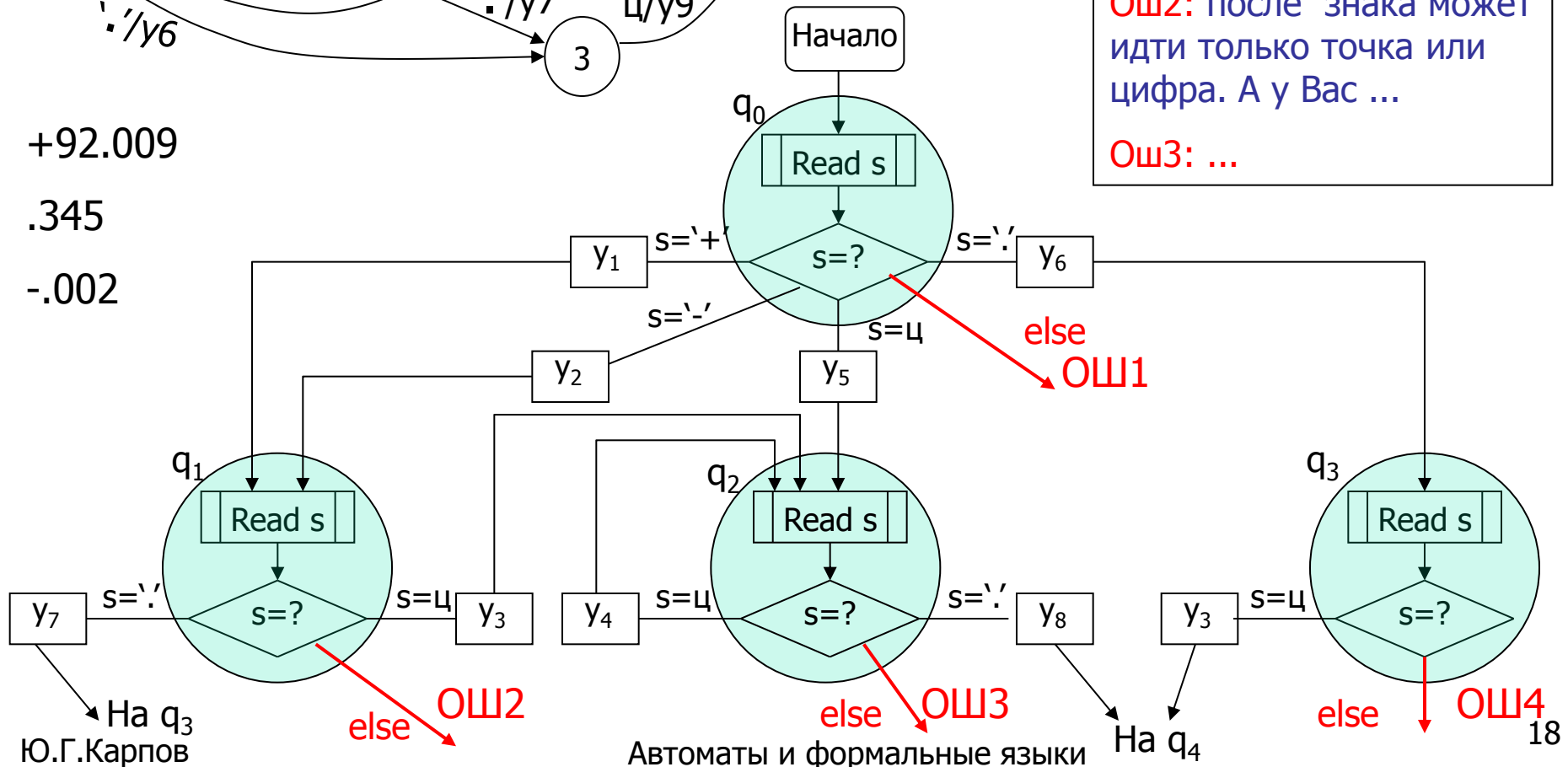
Ош2: после знака может идти только точка или цифра. А у Вас ...

Ош3: ...

+92.009

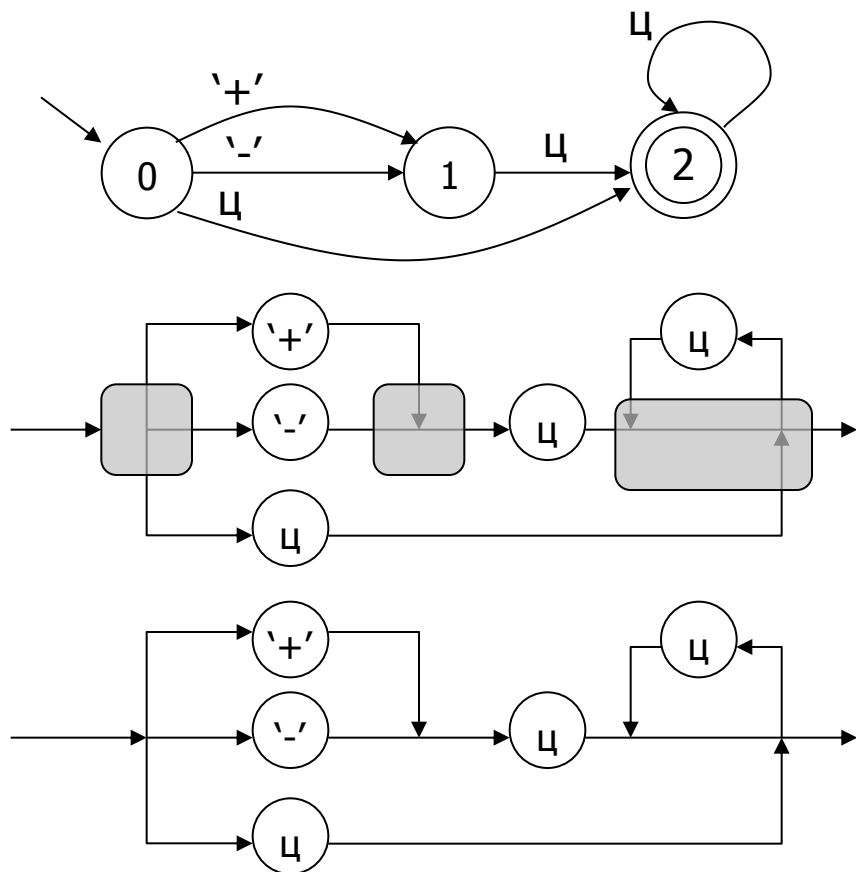
.345

-.002



Конечные автоматы и синтаксические диаграммы

■ Распознавание языка целых чисел



Синтаксическая диаграмма – это направленный граф с одним входом и одним выходом. Вершины графа помечены символами, которые могут встретиться в цепочке языка

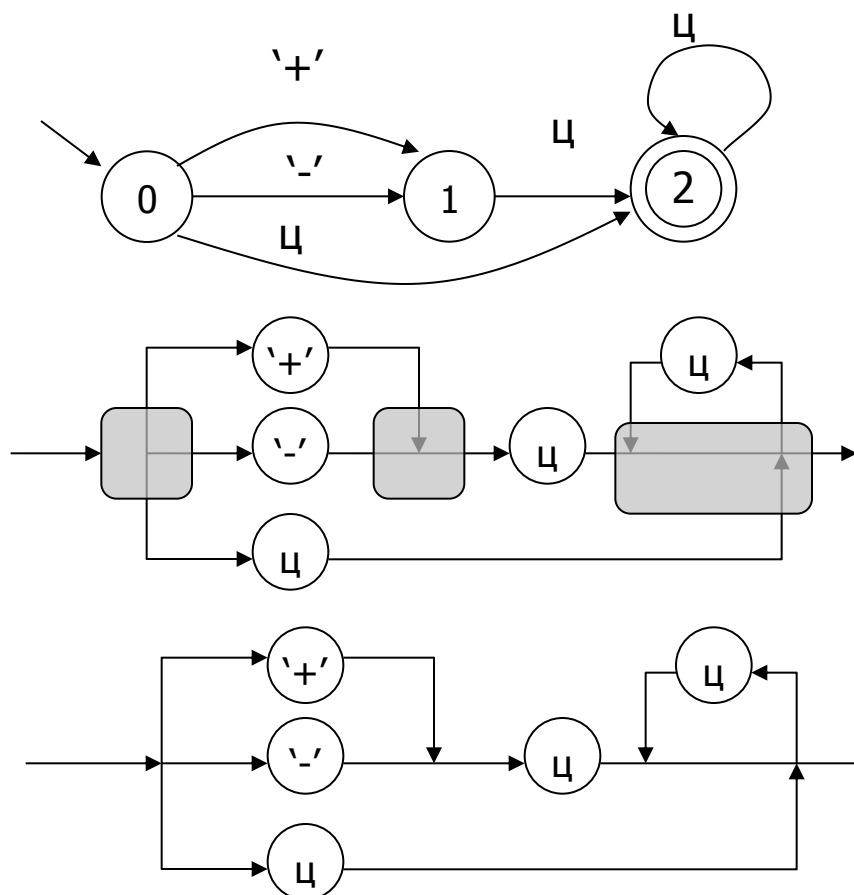
Последовательность символов, встреченных на любом пути от входа к выходу синтаксической диаграммы является цепочкой определяемого этой диаграммой языка

Реализация синтаксической диаграммы выполняется аналогично реализации конечного автомата

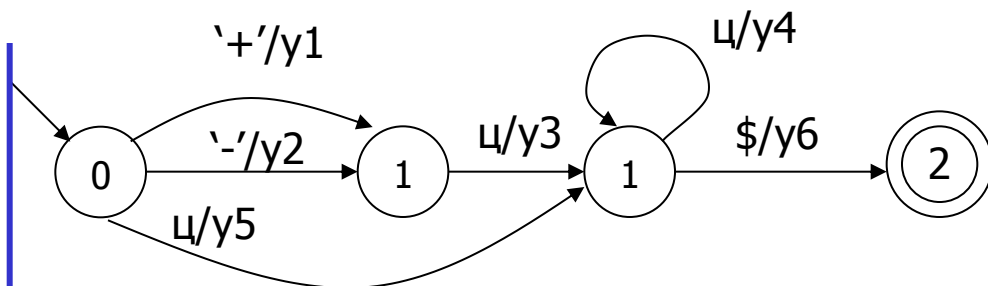
Синтаксическая диаграмма – удобный способ задания языка. Синтаксическую диаграмму можно рассматривать, как порождающий механизм цепочек языка, а иногда (если выбор пути однозначен) - как распознающий механизм

Конечные автоматы и синтаксические диаграммы

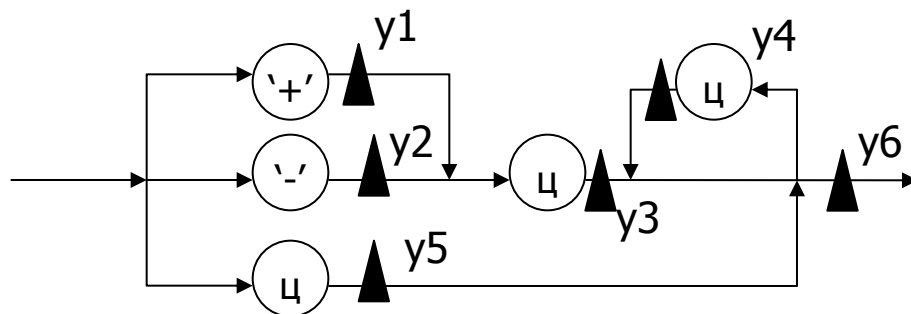
Трансляция языка целых чисел



$\varphi(ц)$ – по конкретной цифре $ц$ выдается двоичное число, изображаемое этой цифрой



Семантические действия встраиваются в диаграмму



$y1: x:=0; z:=1;$

$y2: x:=0; z:=-1;$

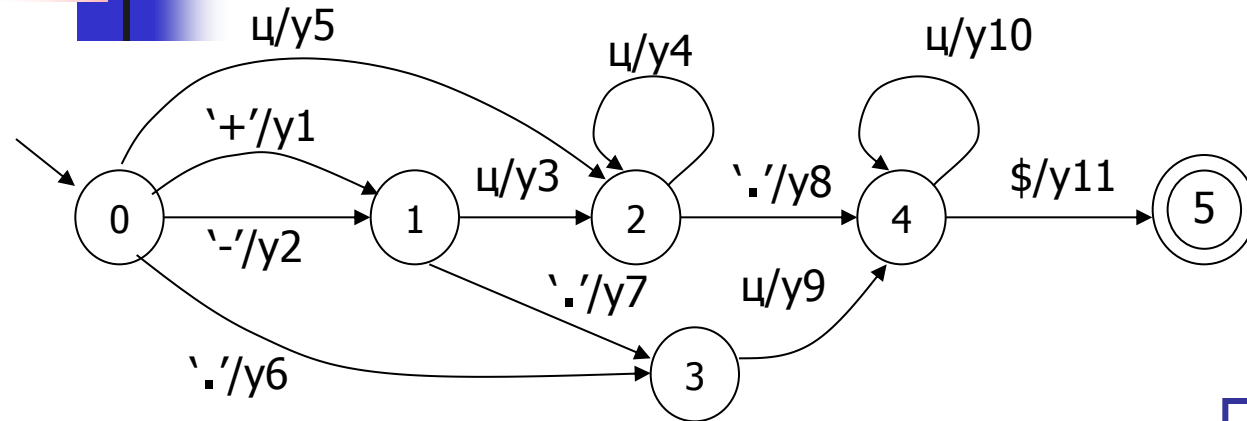
$y3: x:=\varphi(ц);$

$y4: x:=\varphi(ц)+10*x;$

$y5: x:=\varphi(ц); z:=1;$

$y6: X:=x * z;$

Синтаксическая диаграмма и транслятор языка вещественных чисел



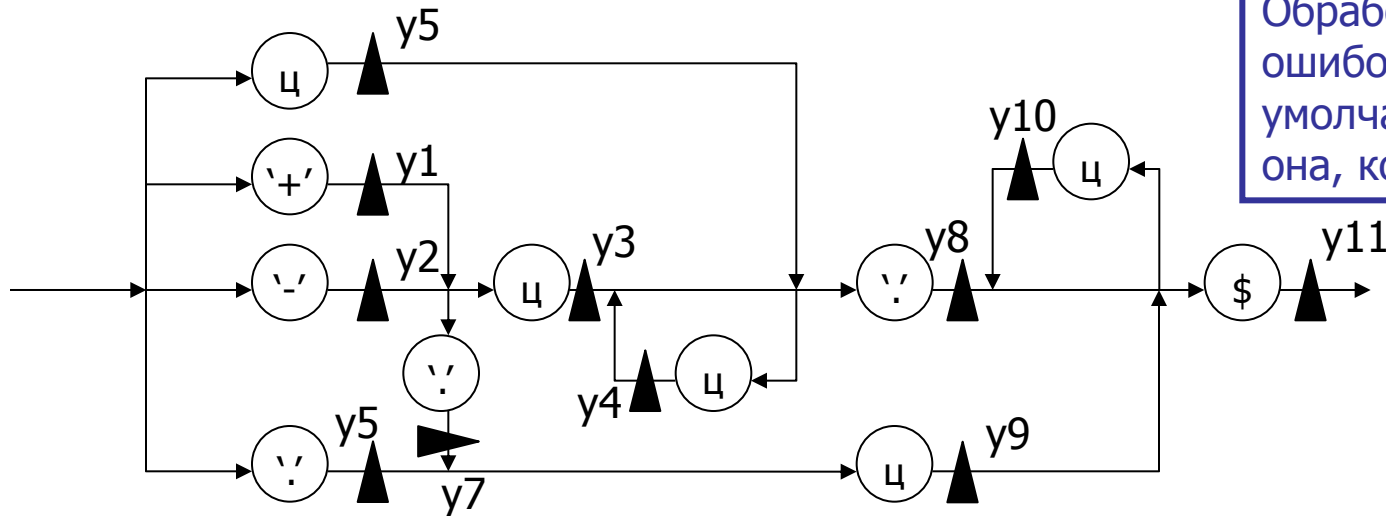
+92.009

.345

-.002

транслятор имитирует
работу распознающего КА

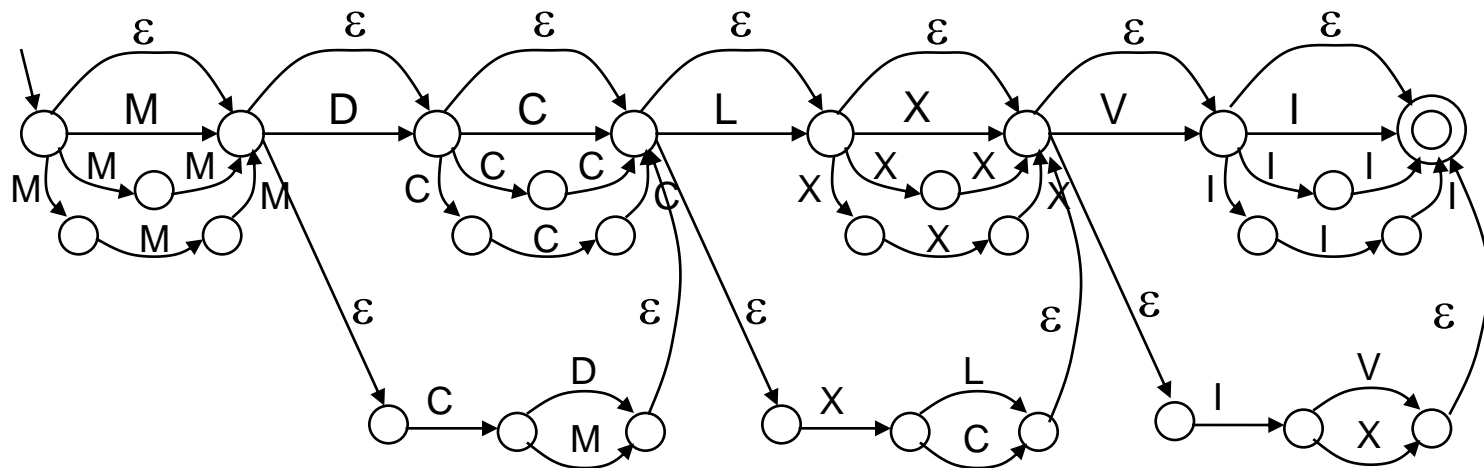
Обработка синтаксических ошибок выполняется по умолчанию. Но в реализации она, конечно, присутствует!!



По цепочке + 3 . 0 5 \$ будет выдана последовательность действий (операций) y1 y3 y8 y10 y10 y11. Фактически, при трансляции входная цепочка сама управляет генерацией последовательности семантических действий

Язык чисел римской системы счисления

XXIV – 24
XLIII – 43
MMIX – 2009

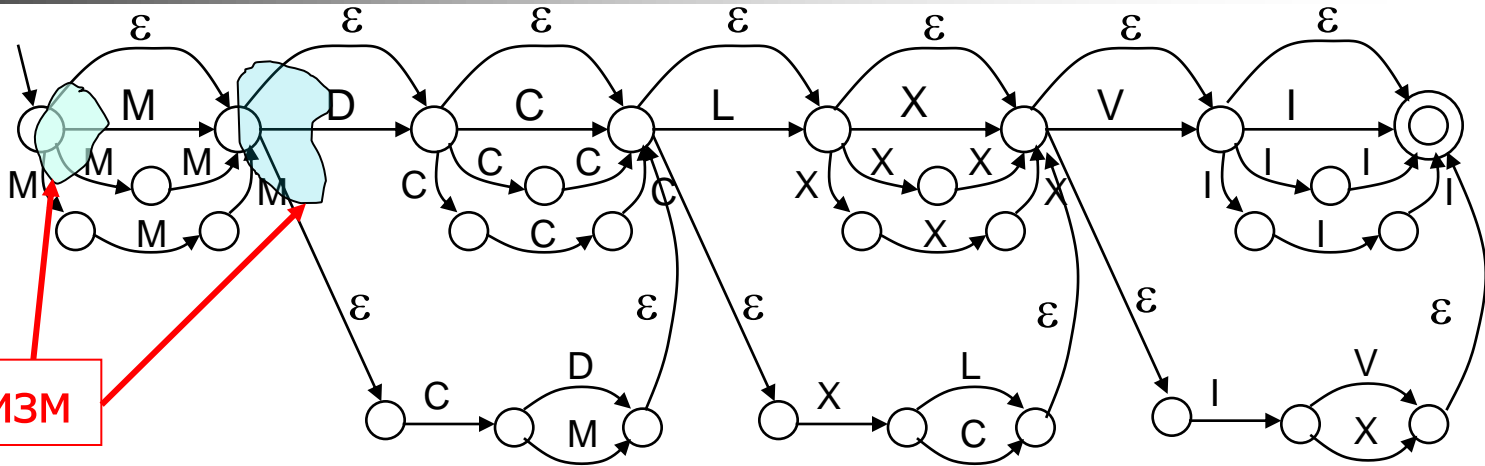


Для построения транслятора нужно построить
ДЕТЕРМИНИРОВАННЫЙ распознаватель языка,
а потом определить подходящие семантические правила

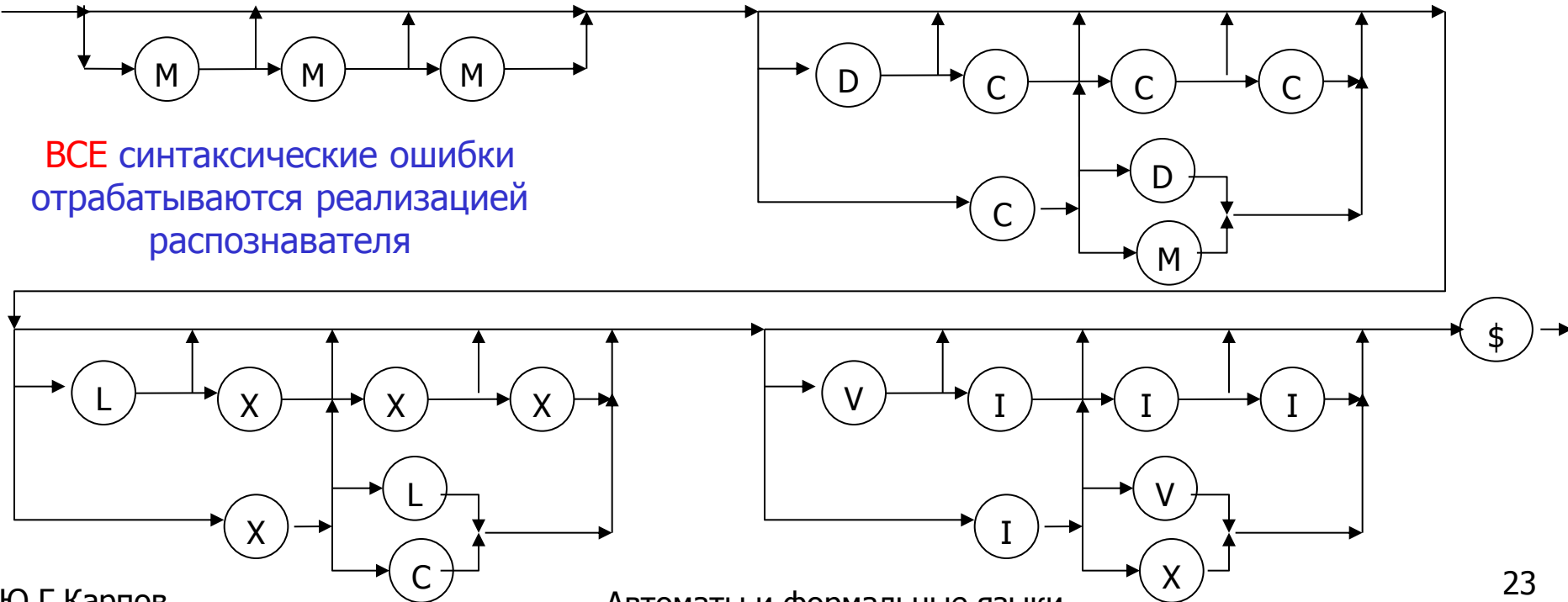
Распознаватель языка римской системы счисления – детерминированный распознаватель

I = 1 V = 5
X = 10 L = 50
C = 100 D = 500
M = 1000

Недетерминизм

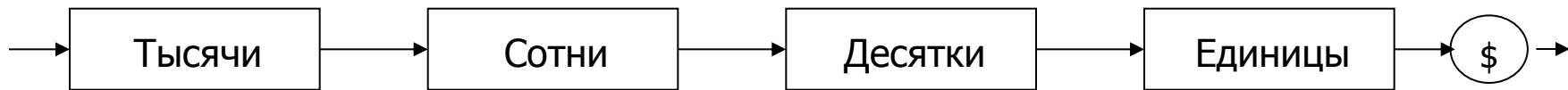


ВСЕ синтаксические ошибки
обрабатываются реализацией
распознавателя

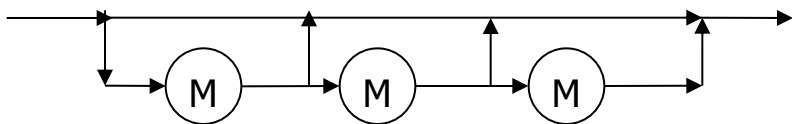


Распознаватель языка чисел римской системы счисления как совокупность модулей

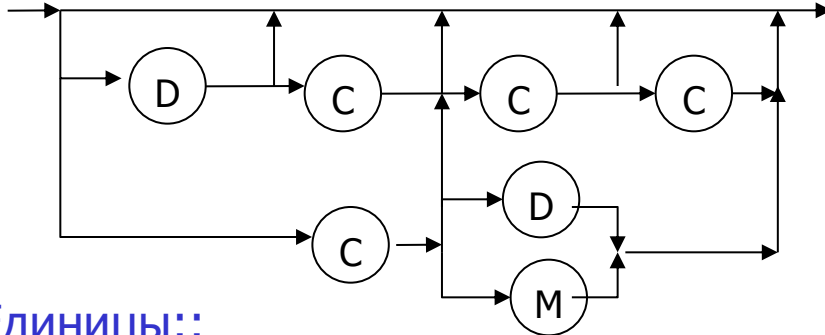
Число::



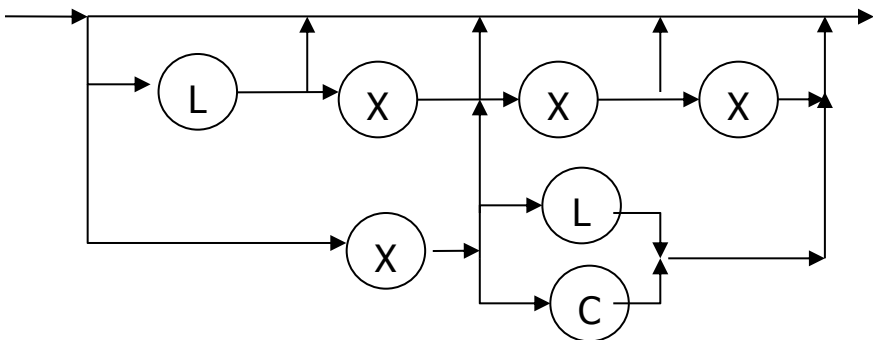
Тысячи::



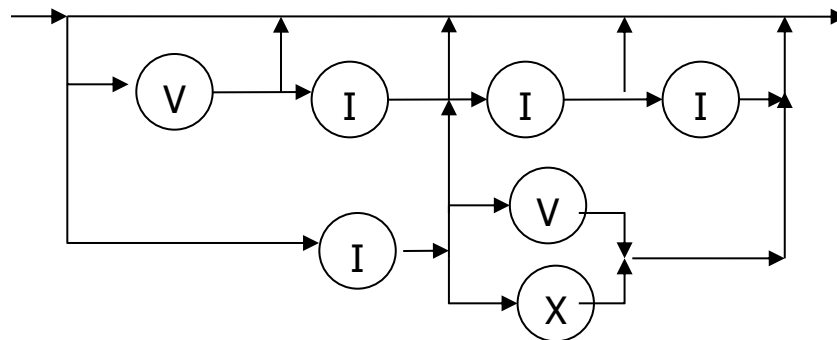
Сотни::



Десятки::



Единицы::



Распознаватель может быть построен из 4-х связанных модулей – процедур распознавания. Каждый модуль распознает одну конструкцию языка

Транслятор языка чисел римской системы счисления из нескольких последовательных модулей

I = 1 V = 5
X = 10 L = 50
C = 100 D = 500
M = 1000

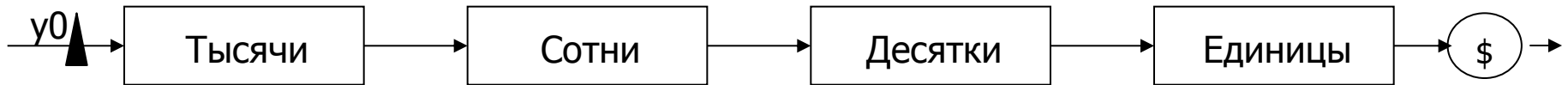
y0: Ч := 0;
y1: Ч += 1000;

y2: Ч += 500;
y3: Ч += 100;
y4: Ч += 300;
y5: Ч += 800;

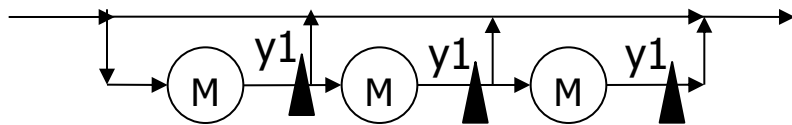
y6: Ч += 50;
y7: Ч += 10;
y8: Ч += 30;
y9: Ч += 80;

y10: Ч += 5;
y11: Ч += 1;
y12: Ч += 3;
y13: Ч += 8;

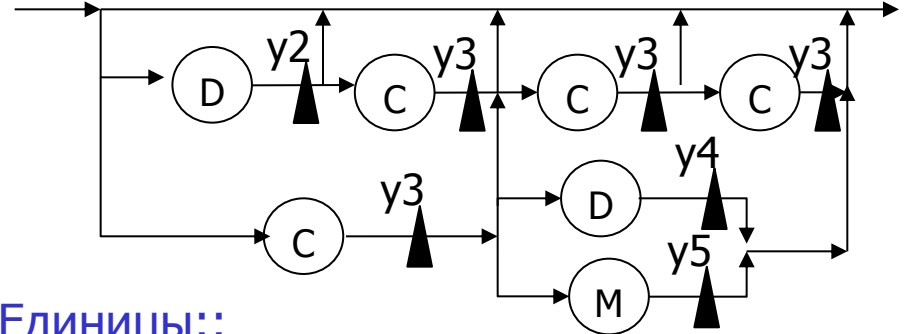
Число::



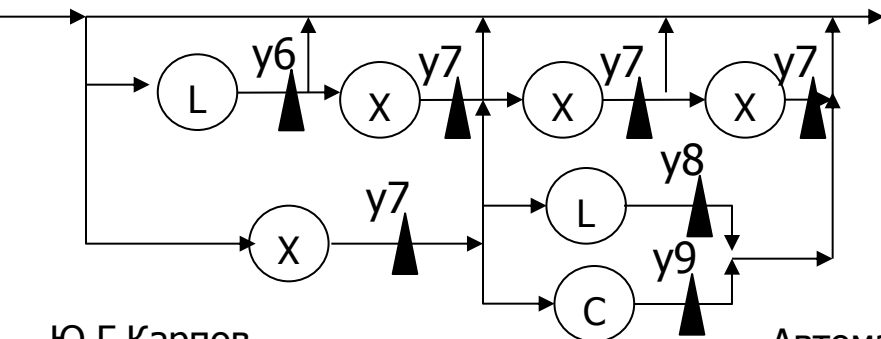
Тысячи::



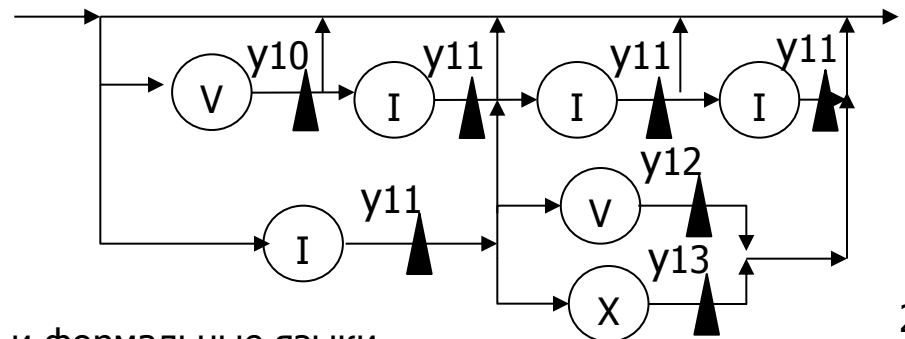
Сотни::



Десятки::



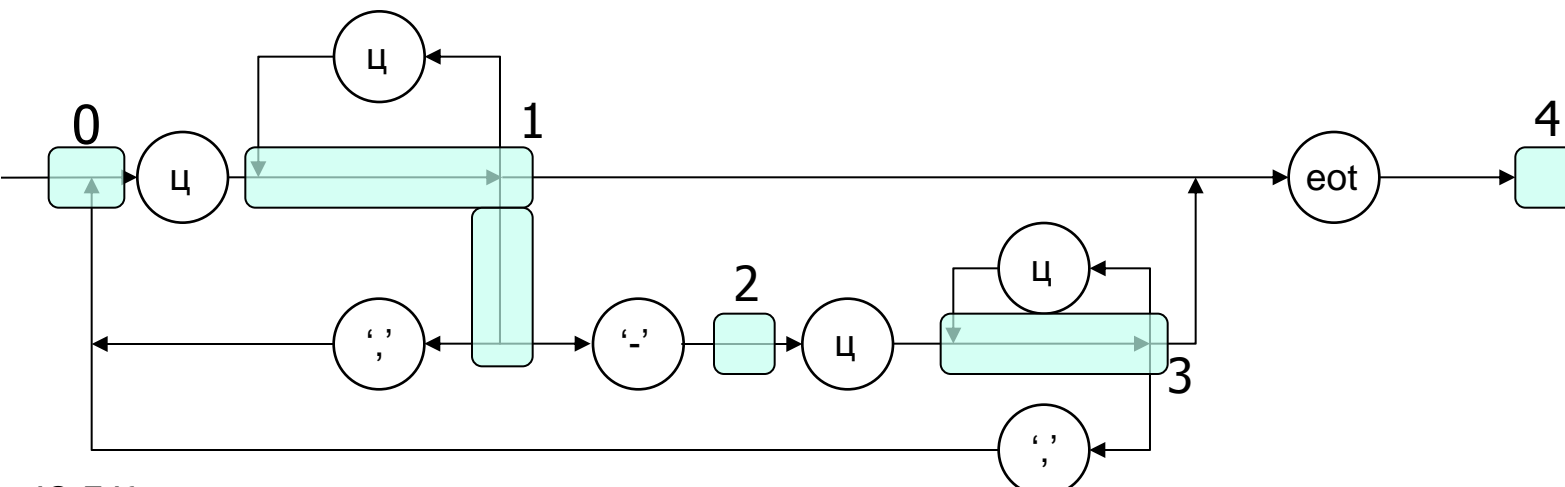
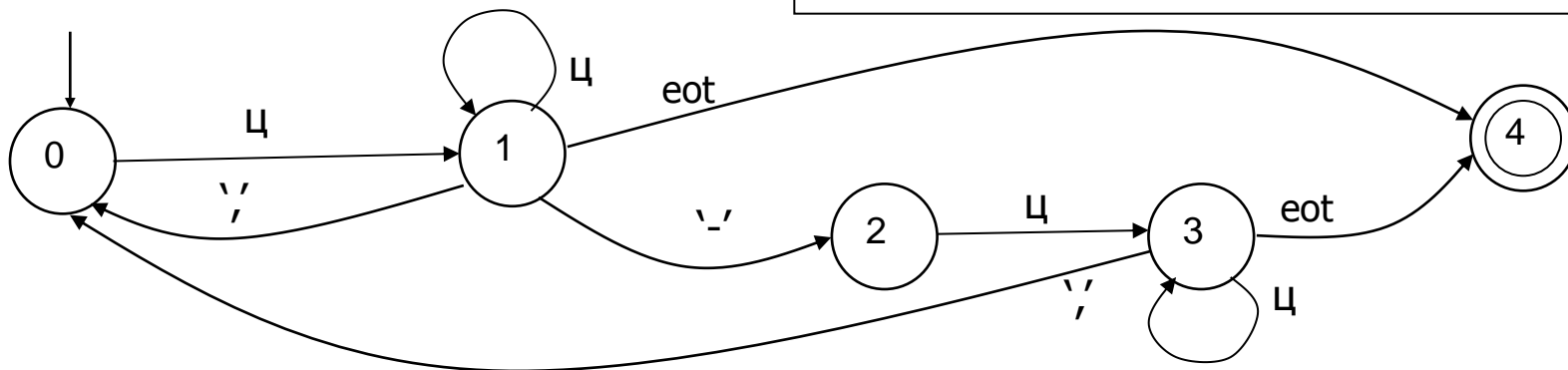
Единицы::



Язык перечисления номеров страниц

- В Microsoft Word печать страниц производится на основе записи пользователя

☐ All
☐ Current page
☒ Pages: **23, 13-54, 124**
 Enter page numbers and/or page ranges separated by commas. For example, 1, 3, 5-12



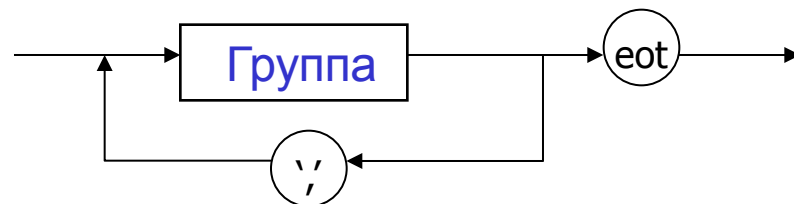
Язык перечисления номеров страниц (2)

Pages: 23, 13-54, 128-97

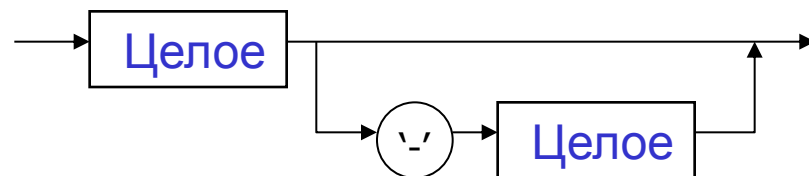
Попробуем по-другому.

Выделим конструкции и для них построим синтаксические диаграммы:

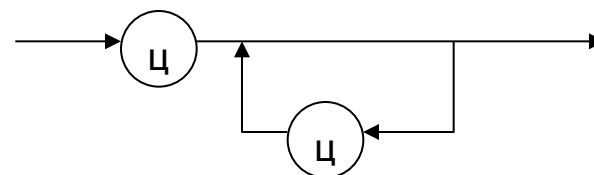
Любая цепочка – это последовательность групп, разделенных запятыми



Любая Группа – это либо одно целое, либо два целых, разделенных тире



Любое Целое – это одна или много цифр



Язык перечисления номеров страниц (3)

Язык можно задать набором синтаксических диаграмм

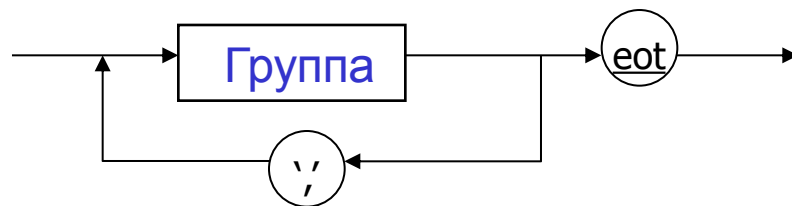
Каждая синтаксическая диаграмма определяет некоторую конструкцию языка

Синтаксическая диаграмма может содержать как терминальные (входные) символы, так и конструкции

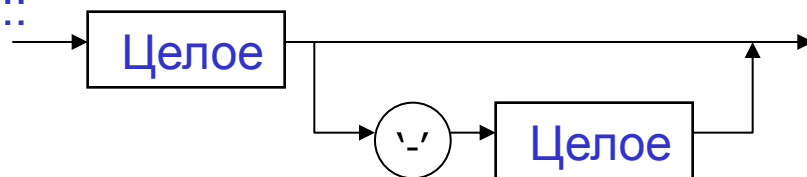
Pages: 23, 13-54, 128-97

Для построения распознавателя можно либо определение каждой вспомогательной конструкции вставить вместо ее имени, а можно строить транслятор, разрабатывая свою программную процедуру для каждой синтаксической диаграммы (каждой конструкции)

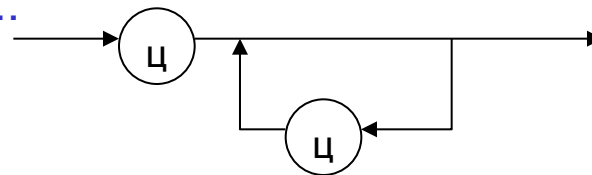
main::



Группа::

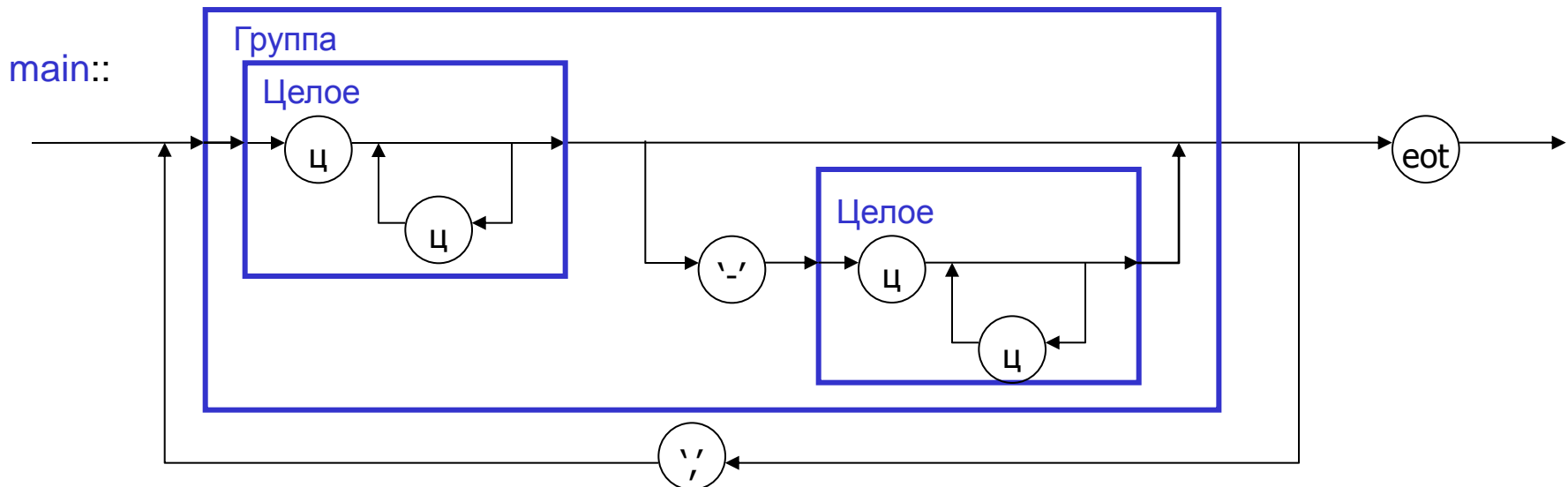
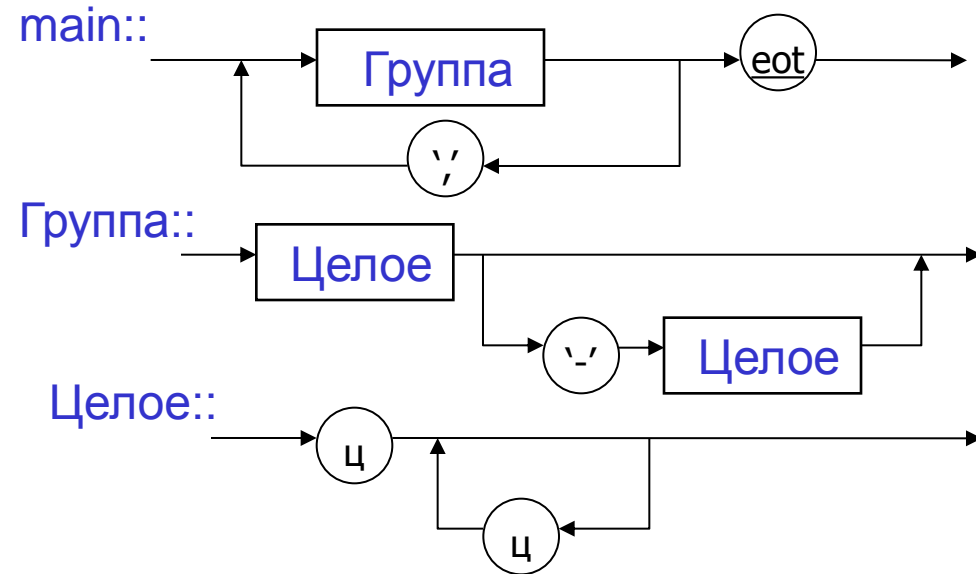


Целое::



Язык перечисления номеров страниц (4)

Результирующая синтаксическая диаграмма для автоматного языка будет содержать только терминальные символы. В случае ее детерминированности распознаватель может быть реализован программно, аналогично реализации детерминированного конечного автомата



Распознаватель языка перечисления номеров страниц

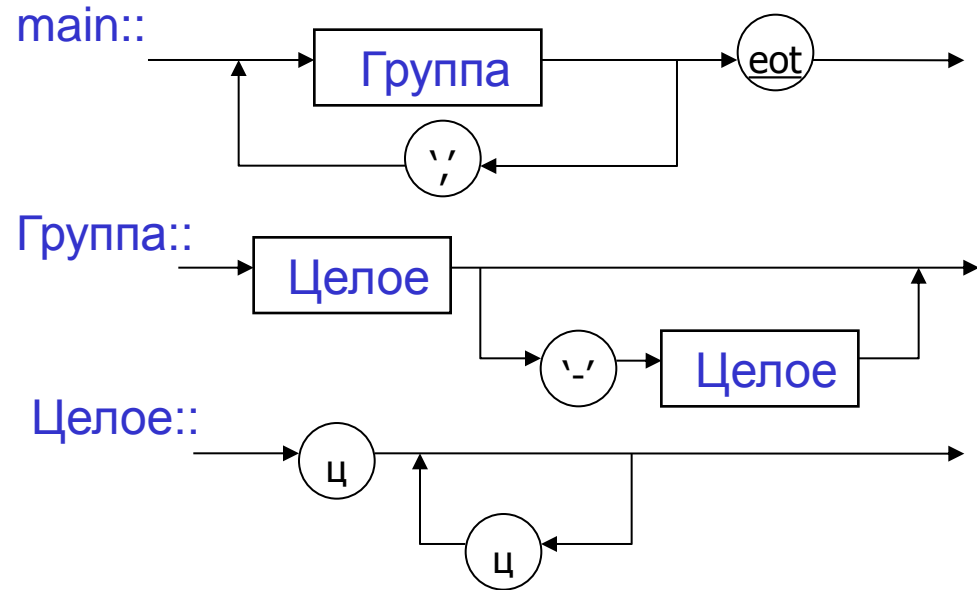
```

int Ук:=0; // указат на очередной символ

main( ) {
m: Группа( );
  if( s[Ук] == 'eot') then { Ук++; break };
  else if( s[Ук] == ',') then { Ук++; goto m };
  else Error1( );
};

Группа( ) {
  Целое( );
  if( s[Ук] == '-') then { Ук++; Целое( ) };
  else skip
};

Целое( ) {
  if( s[Ук] == 'ц') then Ук++;
  else Error2( );
m: if( s[Ук] == 'ц') then { Ук++; goto m };
  else skip
};
    
```



Распознаватель строится как совокупность процедур, каждая из которых распознает свою конструкцию (передвигая указатель Ук)

Перед входом в каждую процедуру Ук должен быть установлен на начало подцепочки, соответствующей конструкции. По выходе из процедуры общий указатель Ук установлен на первый символ ПОСЛЕ этой конструкции

Вопрос: проработал распознаватель – ну и что?? Какой результат?

Транслятор языка перечисления номеров страниц

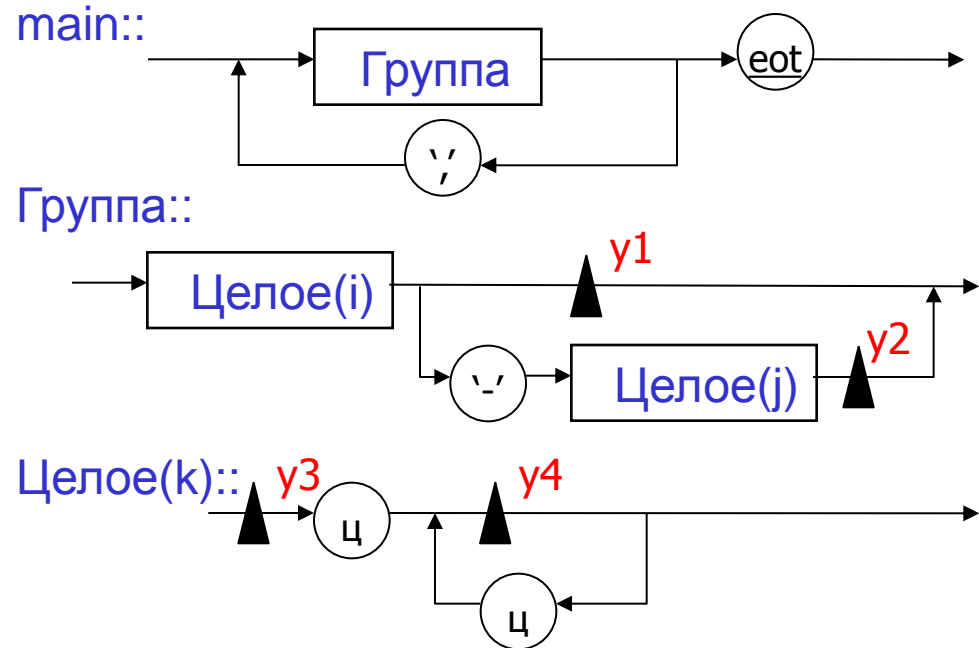
```

int Yk:=0;

main( ) {
  m: Группа( );
  if( s[Yk] == 'eot' ) then { Yk++; break };
  else if( s[Yk] == ',' ) then { Yk++; goto m };
  else Error1( );
};

Группа( ) { int i, j ;
  Целое( i );
  if( s[Yk] == '-' ) then { Yk++; Целое(j); y2 };
  else { y1; skip };
};

Целое( int k ) { y3;
  if( s[Yk] == 'ц' ) then Yk++; y4;
  else Error2( );
  m: if( s[Yk] == 'ц' ) then { Yk++; goto m };
  else skip
};
  
```



Семантики:

y1: Выдача страницы i на печать

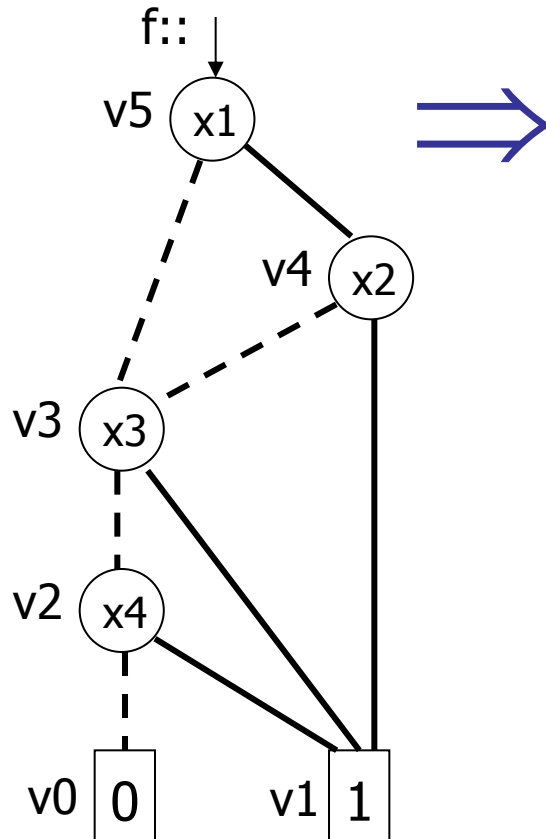
y2: Выдача на печать от страницы i до страницы j (можно вставить проверку, возрастают ли номера страниц, $i \leq j$)

y3: $k:=0$

y4: $k:=k*10 + \varphi(\text{ц})$

Трансляция BDD в программу вычисления значений

- Как построить транслятор из описания BDD в программу вычисления значений логической функции?



Эквивалентная программа

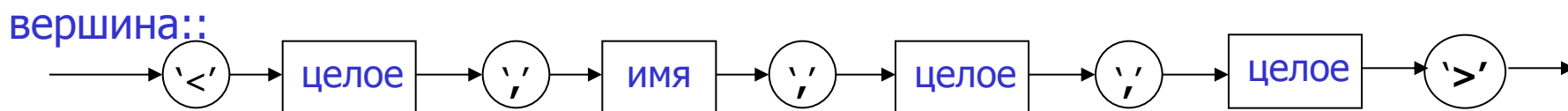
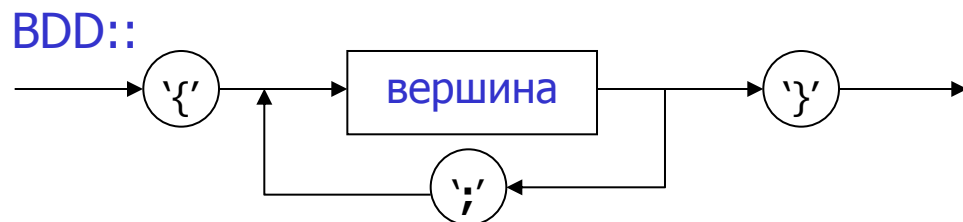
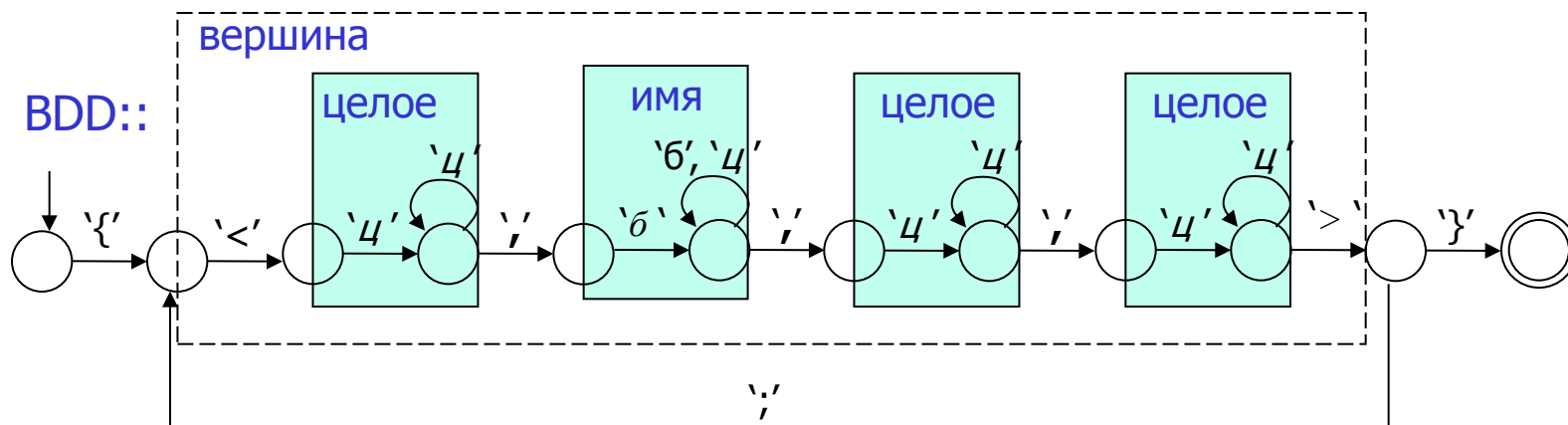
```
v5: if(x1 == 0) goto v3;
    else goto v4;
v4: if(x2 == 0) goto v1;
    else goto v4;
v3: if(x3 == 0) goto v2;
    else goto v1;
v2: if(x4 == 0) goto v0;
    else goto v1;
v1: return (1);
v0: return (0);
```

Будем использовать текстовое представление BDD:

BDD:: f={<5, x1, 3, 4>; <4, x2, 3, 1>; <3, x3, 2, 1>; <2, x4, 0, 1>}

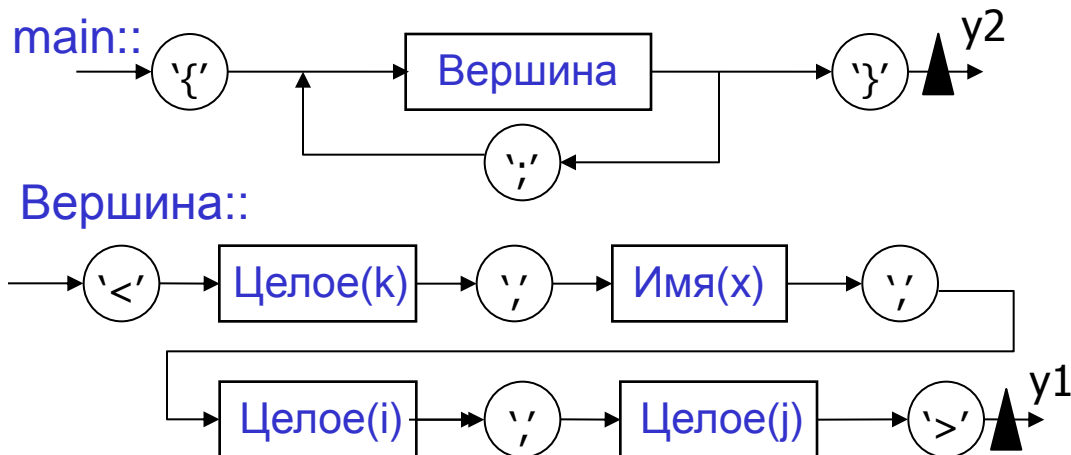
От распознающего автомата к синтаксическим диаграммам

BDD:: $f = \{ \langle 5, x_1, 3, 2 \rangle; \langle 2, x_2, 3, 1 \rangle; \langle 3, x_3, 4, 1 \rangle; \langle 4, x_4, 0, 1 \rangle \}$



Трансляция BDD в программу вычисления значений

BDD:: $f = \{ \langle 5, x_1, 3, 2 \rangle; \langle 2, x_2, 3, 1 \rangle; \langle 3, x_3, 4, 1 \rangle; \langle 4, x_4, 0, 1 \rangle \}$



Целое(k):: ...

Имя (a):: ...

```
v5: if(x1 == 0) goto v3;  
    else goto v4;  
v4: if(x2 == 0) goto v1;  
    else goto v4;  
v3: if(x3 == 0) goto v2;  
    else goto v1;  
v2: if(x4 == 0) goto v0;  
    else goto v1;  
v1: return (1);  
v0: return (0);
```

Семантика — всего две семантических функции (кроме Целого и Имени) генерации текстов:

y1: Ген(` v_k: if(x == 0 goto v_i; else goto v_j; `)

y2: Ген(` v_1: return (1); `); Ген(` v_0: return (0); `)



Язык систем линейных алгебраических уравнений

- Пример программы:

Решить систему 3 уравнений:

$$-3.56 X[1] + 13.0 X[3] = 0.342;$$

$$0.236 X[2] + 223.8 X[3] = -233.3;$$

$$8.12 X[1] - 6.6 X[2] = 5.1$$

методом Гаусса.

Как задать все возможные программы?

Как описать все возможные системы уравнений?

Грамматикой!

Грамматика языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:

$$-3.56 X[1] + 13.0 X[3] = 0.342;$$

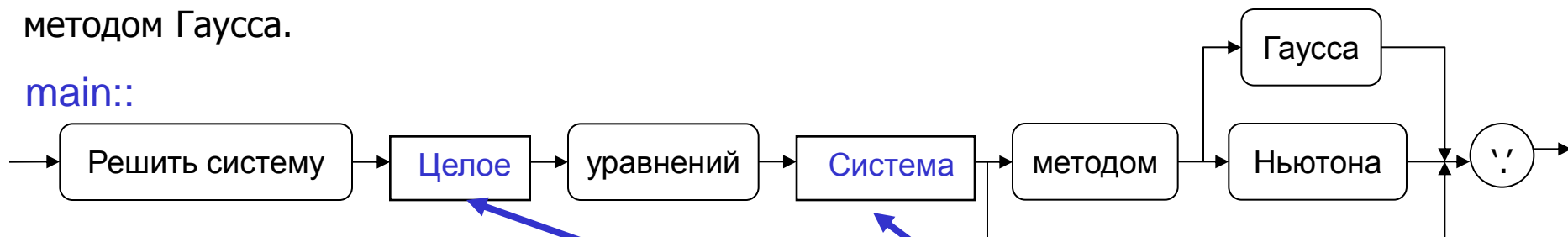
$$0.236 X[2] + 223.8 X[3] = -233.3;$$

$$8.12 X[1] - 6.6 X[2] = 5.1$$

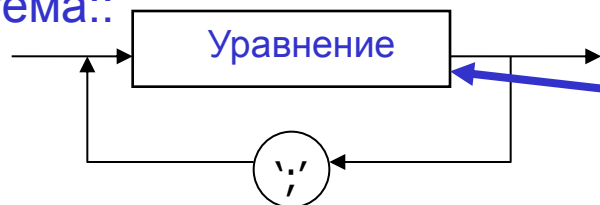
методом Гаусса.

как построить грамматику,
задающую все возможные
правильные тексты такого вида?

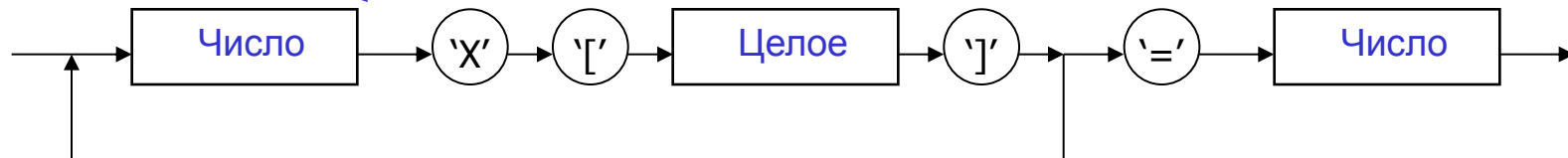
main::



Система::



Уравнение::



Число:: ...

Целое:: ...

Конструкции языка -
подязыки

Распознаватель языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:

$$-3.56 X[1] + 13.0 X[3] = 0.342;$$

$$0.236 X[2] + 223.8 X[3] = -233.3;$$

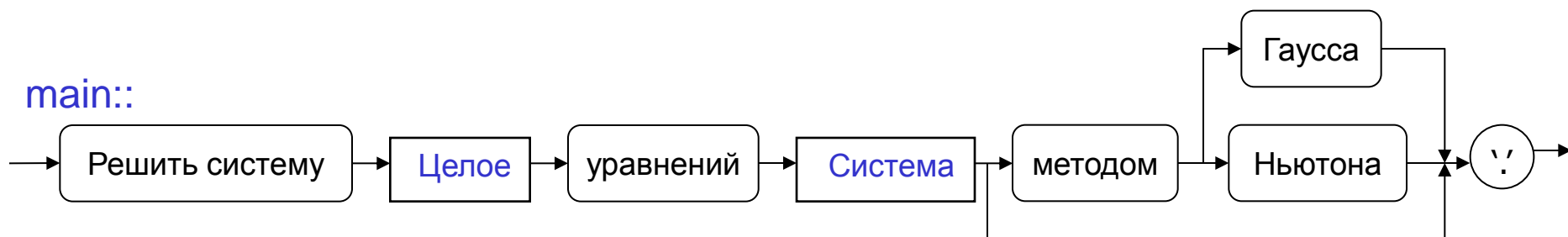
$$8.12 X[1] - 6.6 X[2] = 5.1$$

методом Гаусса.

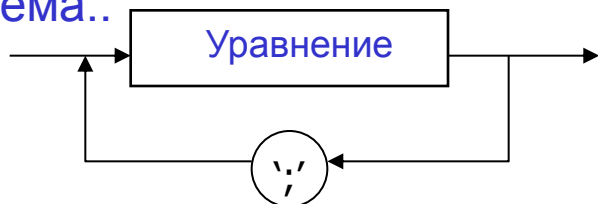
Распознаватель строится однозначно по синтаксическим диаграммам не всегда!

Только тогда, когда выбор в каждом разветвлении синтаксической диаграммы однозначен

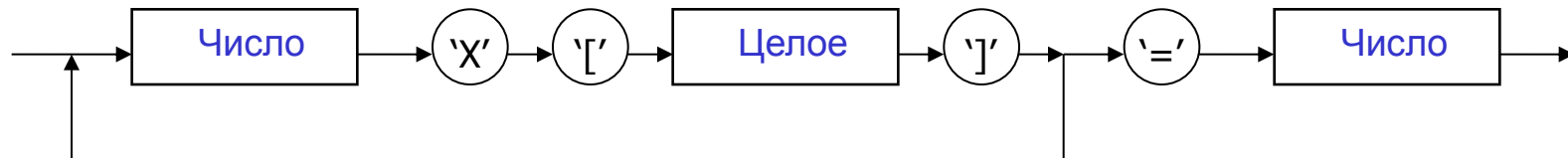
main::



Система::



Уравнение::



Число:: ...

Целое:: ...

Распознаватель языка систем линейных алгебраических уравнений

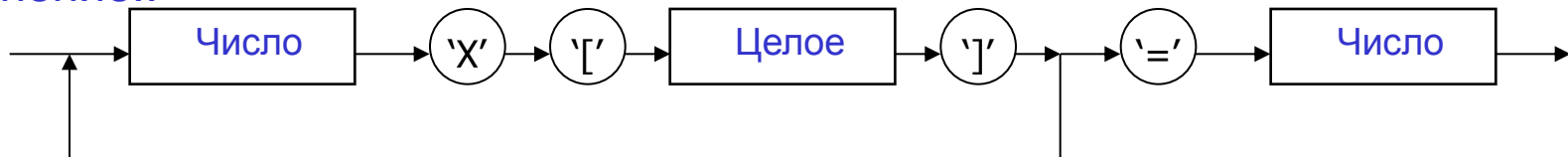
```
int Yk:=0;
```

```
main( ) {  
  if( s[ Yk ] <> 'Решить систему') then Ош(1) ;  
    else Yk +=16; Целое( );  
  if( s[ Yk ] <> 'уравнений:') then Ош(2) ;  
    else Yk +=10; Система( ); ...  
};
```

```
Система( ) {  
  m: Уравнение( );  
  if( s[Yk] == ';' ) then { Yk++; goto m }; else skip  
};
```

```
Уравнение( ) {  
  m: Число( );  
  if( s[Yk] == 'X' ) then Yk++; else Ош3( ); );  
  if( s[Yk] == '[' ) then Yk++; else Ош4( ); ;  
  Целое( );  
  if( s[Yk] == ']' ) then Yk++; else Ош5( );  
  if( s[Yk] <> '=' ) goto m; else {Yk++; Число( ) }  
};
```

Уравнение::



Решить систему 3 уравнений:

$$-3.56 X[1] + 13.0 X[3] = 0.342;$$

$$0.236 X[2] + 223.8 X[3] = -233.3;$$

$$8.12 X[1] - 6.6 X[2] = 5.1$$

методом Гаусса.

Какие ошибки выявит
распознаватель?

Система::



Семантика языка систем линейных алгебраических уравнений

Решить систему 3 уравнений:

$$-3.56 X[1] + 13.0 X[3] = 0.342;$$

$$0.236X[2] + 223.8 X[3] = -23.3;$$

$$8.12 X[1] - 6.6 X[2] = 5.1$$

методом Гаусса.

y1: Объявить матрицы $A[N,N] = 0$, $B[N]$ и $X[N]$;

y2: $k := 1$;

y3: $k++$;

y4: $A[i,j] := a$;

y5: $B[i] := b$;

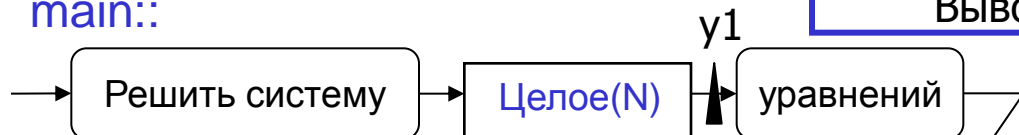
y6: ГАУСС(A, B, N, X, E); (E – тип ошибки)

y7: НЬЮТОН(A, B, N, X, E);

y8: Вывод результата X , если $E=0$;

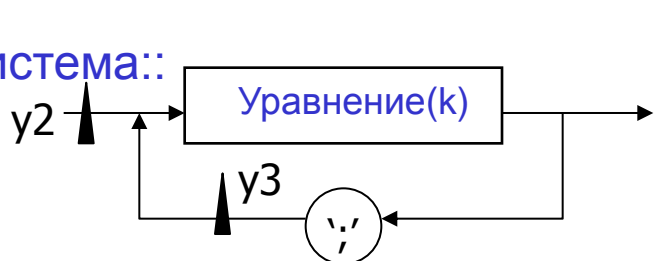
Вывод причины ошибки i , если $E \neq 0$;

main::

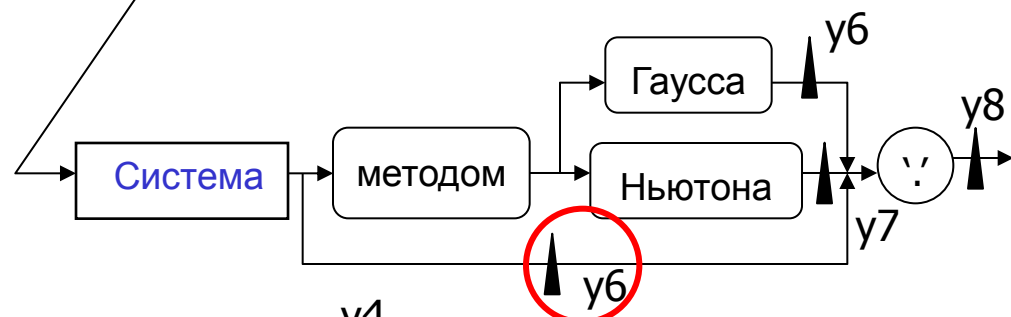
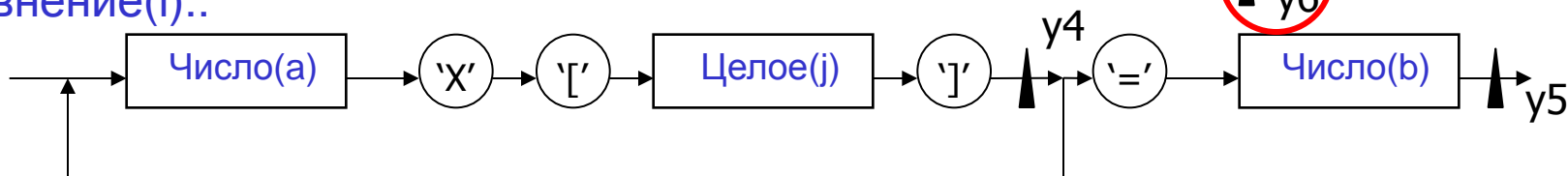


Какие проверки можно сделать??

Система::



Уравнение(i)::



Число(r):: ... Целое(n) :: ...

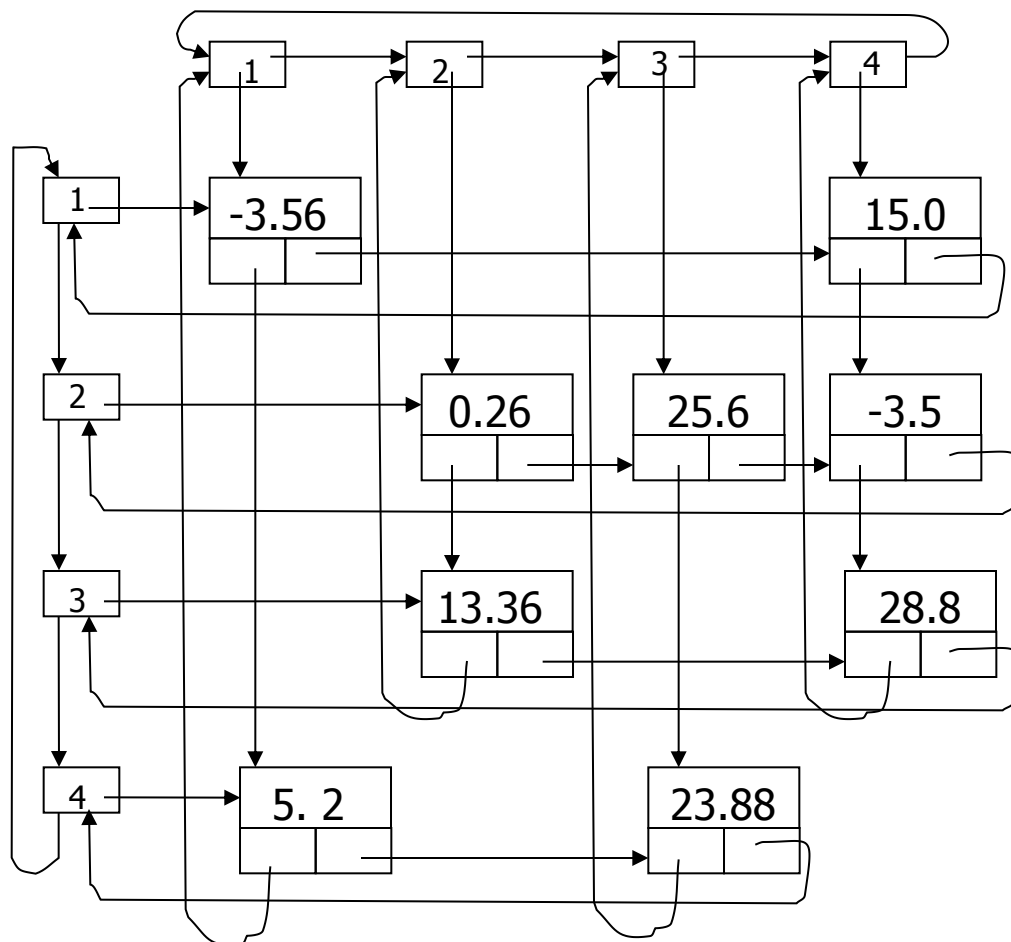
Пример транслятора: Построение структуры данных для разреженной матрицы коэффициентов

Вход – разреженная матрица:

$-3.56 a[1] + 15.0 a[4];$
 $0.26 a[2] + 25.6 a[3] - 3.5 a[4];$
 $13.36 a[2] + 28.8 a[4];$
 $15.2 a[1] + 23.88 a[3];$

Для самостоятельной проработки

Выход – списочная структура



Пример транслятора простого языка присваиваний

■ Пример программы:

begin

x0 := input;

x1 := -23;

x2 := x1*56;

print (x1, x0)

end

В выражении не больше
одной операции

Для самостоятельной проработки

Варианты расширения языка:

- разные типы (int, real) и операции;
- логические переменные и операторы
- условное присваивание;
- логические переменные и операции;
- инкремент и декремент;
- комментарии;
- не только десятичные числа;
- ...

1. Построить транслятор этого языка

2. В качестве дополнительного варианта можно использовать бестиповые переменные: переменная считается объявленной при первом использовании, тип переменной определяется типом присвоенного выражения. Инициализированной переменной можно присваивать значения типов, отличающихся от первоначально присвоенного

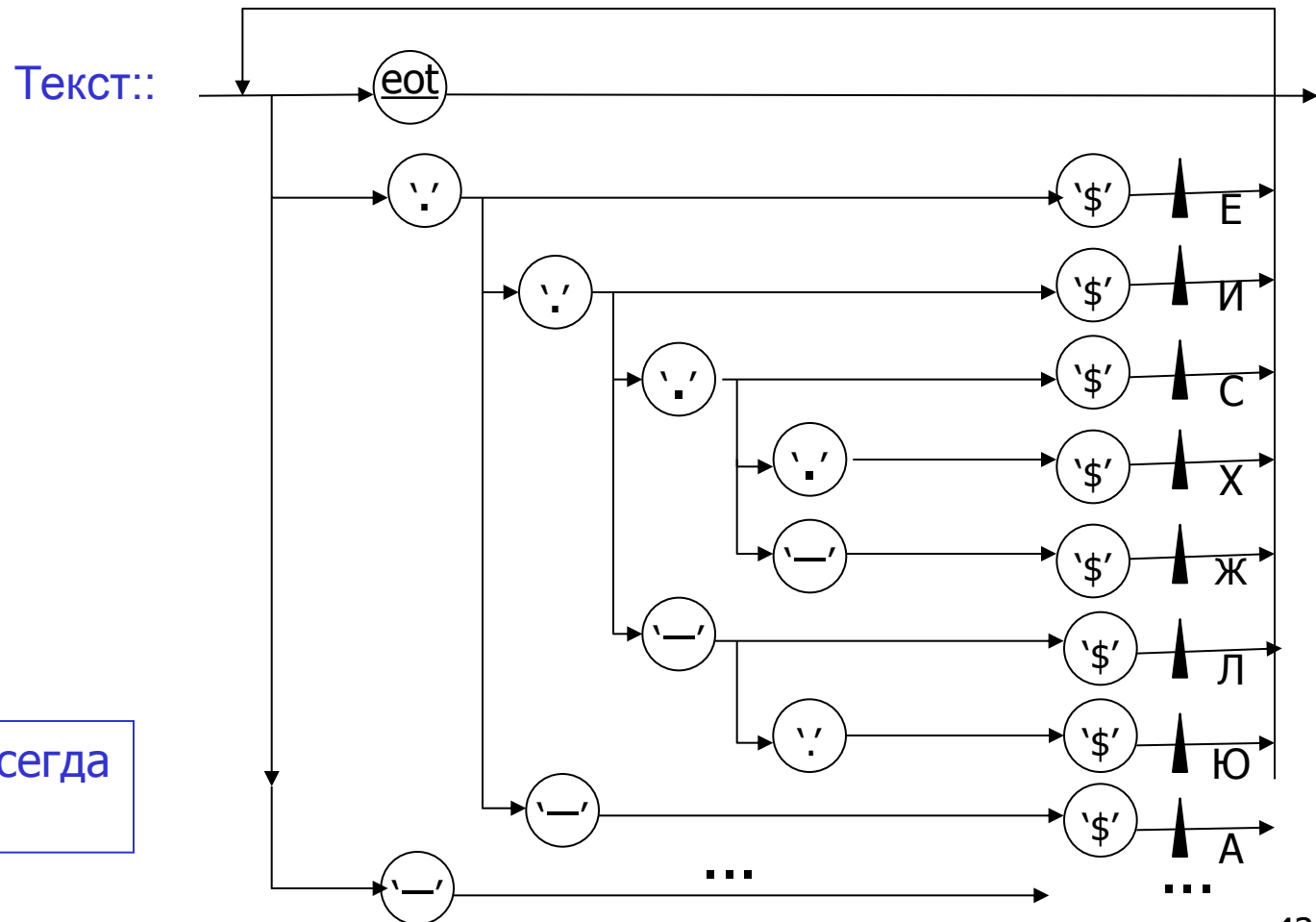
Транслятор языка азбуки Морзе как синтаксическая диаграмма

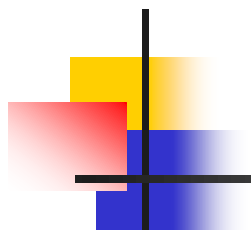
- А · —
- Б — · · ·
- В · — —
- Г — — ·
- Д — · ·
- Е ·
- Ж · · · —
- З — — · ·
- И · ·
- К — · —
- ...

Конечный язык — всегда автоматный!

· · · \$ · — \$ · · · — \$ · — \$ eot

САЖА





Лексический анализ языков программирования

Лексический анализ языков программирования

- Языки программирования высокого уровня не являются автоматными
- Некоторые фрагменты ЯВУ можно описать автоматами: имена, константы...

```
begin
/***** ПРОГРАММА вычисления НОД *****/
  m13 := read;      /* переменная m13 читается из входного потока */
  alpha237:=read;   // переменная alpha237 означает ширину */
  while m13 <> alpha237 do
    if m13 > alpha237 then m13 := m13 - alpha237
    else
      alpha237 := alpha237 - m13;
      write(alpha237)
  end
```

Чем неудобна программа на входном языке?

1. **Бессмысленны отдельные символы**: например, **e** в словах begin, read, или в именах
2. **Все имена** с точки зрения синтаксиса – одно и то же, но имеют разную семантическую хар-ку (так же и все константы)
3. **Каждое служебное слово** представляет единый 'символ'
4. **Пробелы и комментарии** – даже описать включение произвольного их числа в любое место программы формально очень трудно
5. Некоторые группы входных символов представляют одну лексему: ':=', '>=' ...

Программа набирается на клавиатуре в коде ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- Если на клавиатуре наберем:

```
for j := 1 to max do  
  x2[j] := 10;
```

то получим поток байтов:

```
66 6F 72 20 6A 20 3A 3D 20 31 20 64 6F 20 6D 61 78 20 64 6F 0D 0A 09  
20 20 20 78 32 5B 6A 5D 20 3A 3D 20 31 30 3B
```

Назначение лексического анализа

- В реальных трансляторах первой фазой является так называемый лексический анализ входной программы - предварительная обработка входного текста с выделением в нем структурно значимых единиц – лексем.

Лексема – это та минимальная единица языка, которая имеет смысл с точки зрения синтаксиса

- В естественном языке лексемами являются **не буквы, а слова** (словоформы), в языке программирования – не отдельные символы, а имена, служебные слова, константы, знак оператора присваивания из двух символов `:=` и т.д.

- На входе транслятора исходная программа

```
for j := 1 to max do  
  x2[j] := 10;
```

представлена в виде неструктурированного потока байтов в коде ASCII:
66 6F 72 20 6A 20 3A 3D 20 31 20 64 6F 20 6D 61 78 20 64 6F 0D 0A 09 20
20 20 78 32 5B 6A 5D 20 3A 3D 20 31 30 3B

- Символы не имеют смысла, смысл имеют **ЛЕКСЕМЫ** – группы символов

```
for j := 1 to max do x2 [ j ] := 10 ;
```

- В этом фрагменте 14 лексем: служебные слова: for, to, do; 4 лексемы 3-х имен: j, max, x2; 2 вхождения лексемы присваивания := и т.д.

Лексический анализ языков программирования

- Задача лексического анализа – представить исходную программу как последовательность лексем (лексических единиц).
- Лексемы и являются терминальными символами грамматики языка

```
begin
  /***** ПРОГРАММА вычисления НОД *****/
  m13:=read;      // переменная m13 читается из входного потока
  alpha237:=read; /* переменная alpha237 означает ширину */
  while m13 <> alpha237 do
    if m13 > alpha237 then m13:=m13 + alpha237
  else
    alpha237 := alpha237 - m13;
    write(alpha237)
end
```

Вопрос: write(...) – внутри цикла, или нет???

По этой последовательности байтов лексический анализатор должен построить следующую последовательность лексем:

```
begin i0 ass read sc i1 ass read sc while i1 q1 i1 do if i0 q2 i1 then i0 ass i0 addop1 i1
else i1 ass i1 addop0 i0 sc write lb i1 rb end
```

36 лексем

Каждая лексема имеет свой смысл, так же, как при трансляции предложений естественного языка каждое слово в словаре имеет свой смысл

Кодирование лексем при лексическом анализе

```
begin i0 ass read sc i1 ass read sc while i1 q1 i1 do if i0 q2 i1 then i0 ass i0 addop1
i1 else i1 ass i1 addop0 i0 sc write lb i1 rb end
```

begin – служебное слово begin

read – служебное слово read

...

i₀, i₁, ... Имена переменных с номерами 0, 1, ...

ass – assign (присваивание :=)

sc – semicolon (;)

q₀, q₁, q₂, ... – отношения =, ≠ (<>), >, <, ...

addop₀, addop₁ – операция типа сложения (- и +)

...

Лексема:

тип

номер

Кодировка лексем - пример

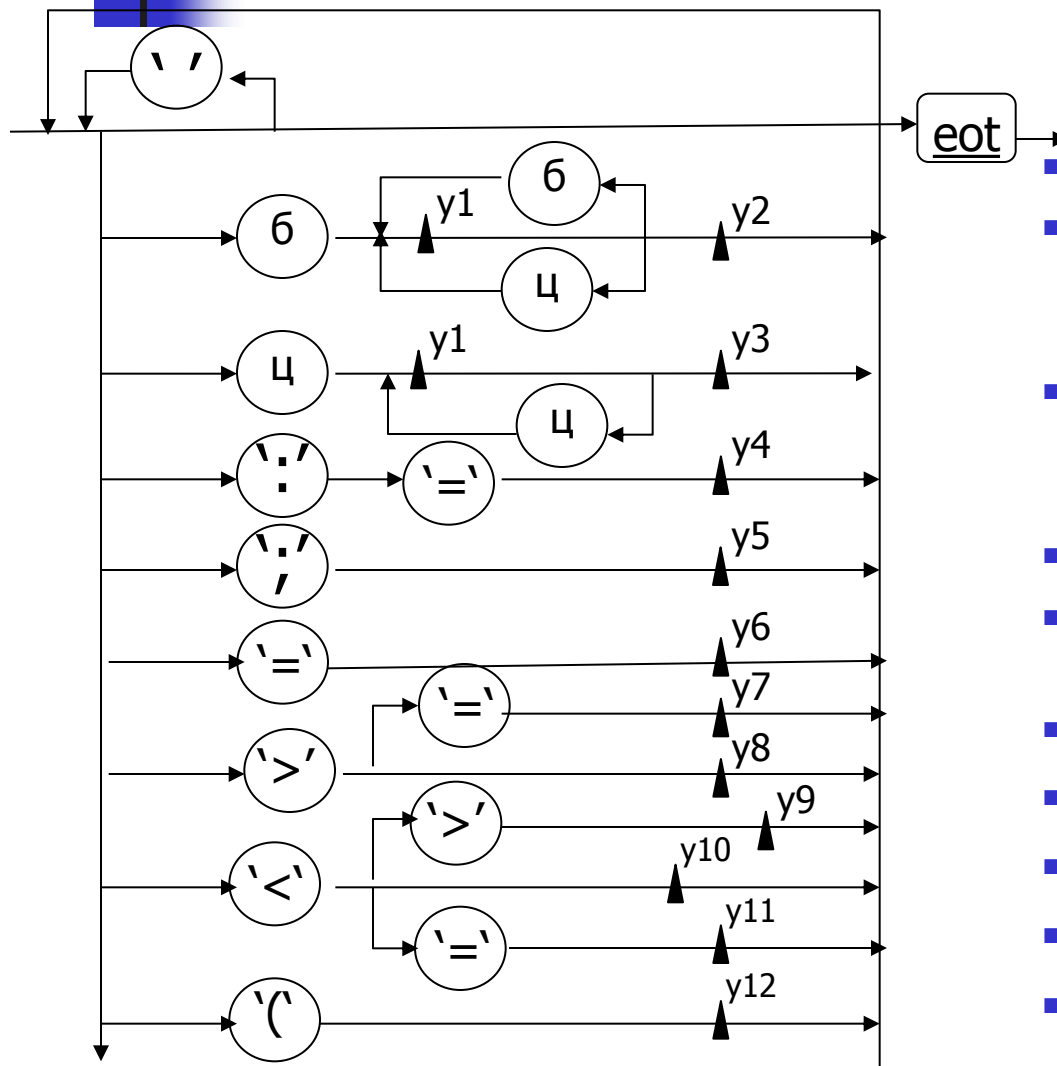
Лексема:

тип

номер

Nтипа	Лексема	Представление	Номер в типе	Пример кодировки
0	begin	<u>begin</u>		<0,0>
1	while	<u>while</u>		<1,0>
...				
21	имена	i_k	k – номер в таблице имен	<21,3>- третье имя
22	конст	c_k	k – номер в таблице констант	<22,5>–пятая константа
23	:=	ass (от assign)		<23,0>
24	;	sc (semicolon)		<25,0>
25	=, <>, >, <	q_k	= k=0, <>k=1, >k=2, < k=3,	<25,2>– отношение '>'
26	-, +	addop _k	- k=0, + k=1	<26,1> – сложение '+'

Лексический анализатор: синтаксическая диаграмма



Лексема:

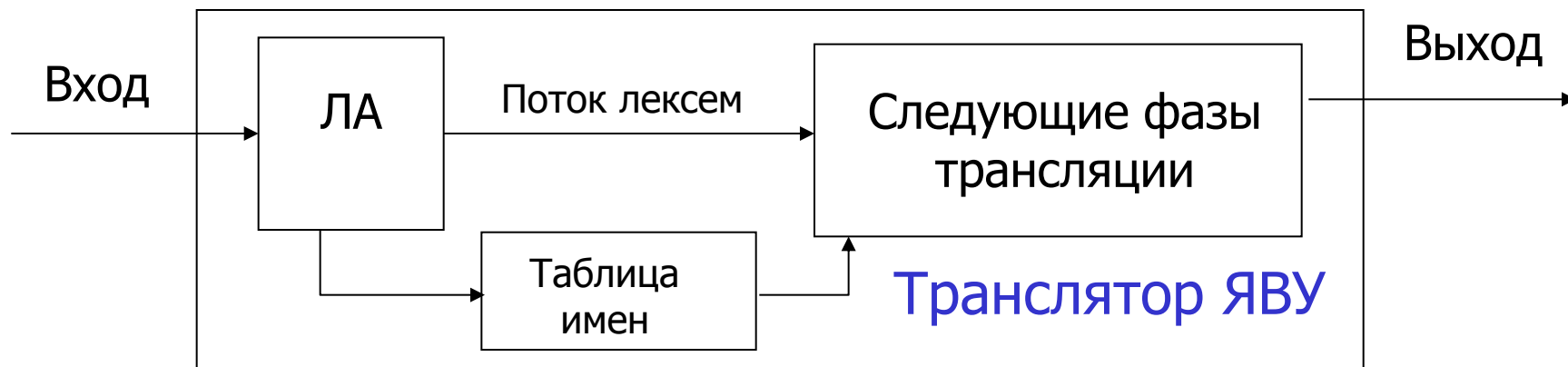
тип	номер
-----	-------

- y1: символ - в буфер
- y2: это служебное слово? Если нет, то проверить/добавить в таблице имен. Выдать соотв. лексему
- y3: проверить/добавить в таблице констант
Выдать лексему <22, k>
- y4: Выдать лексему <23,0> (ass '=')
- y5: Выдать лексему <24,0> (semicolon - ';')
- y6: Выдать лексему <25,0> ('=')
- y7: Выдать лексему <25, 5> ('≥')
- y8: Выдать лексему <25,3> ('>')
- y9: Выдать лексему <25,1> ('≠')
- y10: Выдать лексему <25,2> ('<')
- y11: Выдать лексему <25,4> ('≤')
- y12: Выдать лексему <28,0> ('(')

Лексический анализатор – первый проход транслятора ЯВУ

Лексема:

тип	номер
-----	-------



Вход – цепочка кодов символов клавиатуры:

```
begin m13:=read;alpha237:=read; while m13<>alpha237 do
if m13> alpha237 then...
```

Выход – цепочка кодировок лексем:

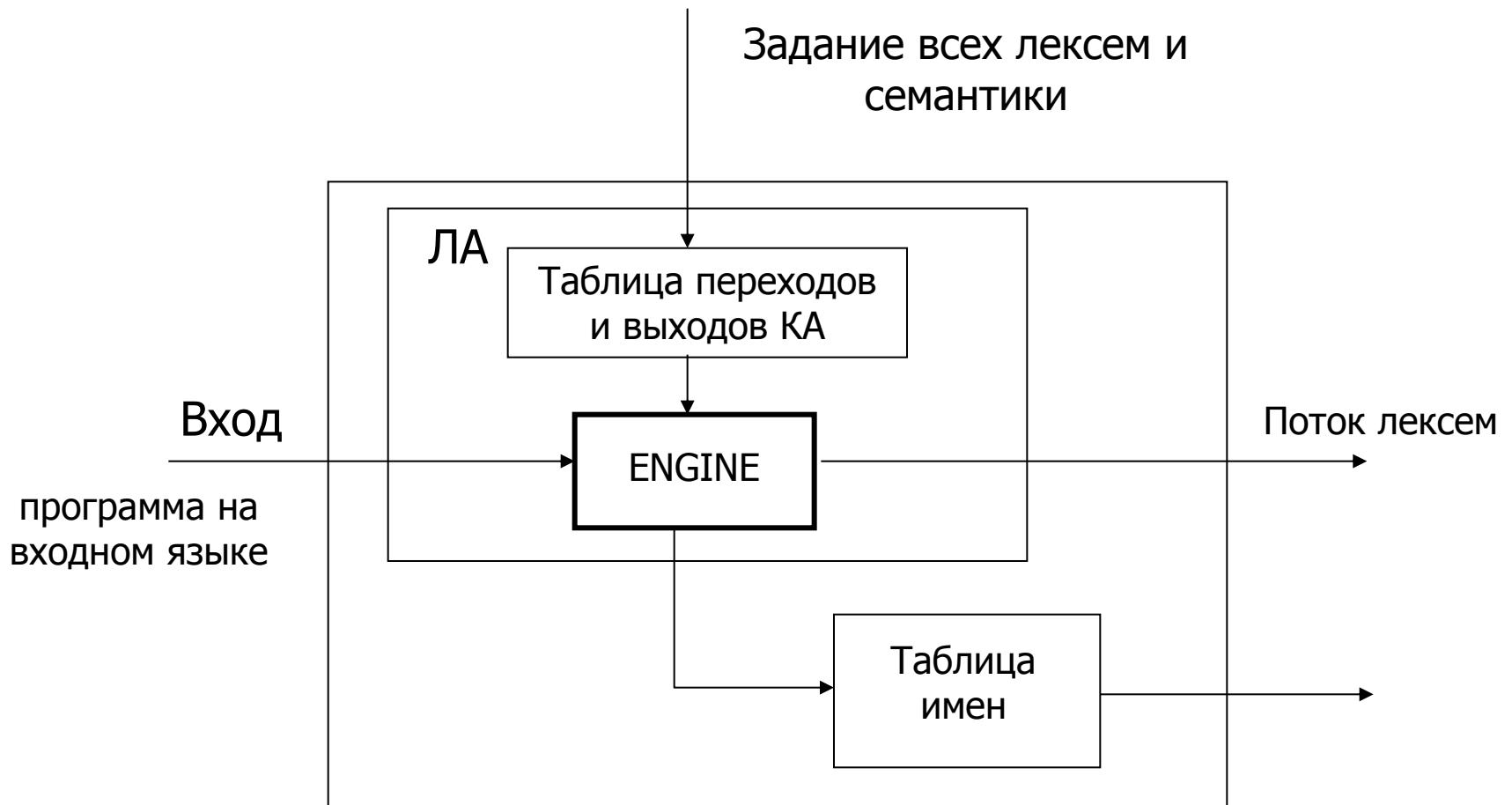
```
begin i0 ass read sc i1 ass read sc while i1 q1 i1 do if i0 q2 i1 then i0 ass i0 addop0 i1
else i1 ass i1 addop0 i0 sc write lb i1 rb end - 36 лексем
```

- Структура таблицы имен:

N	имя	тип
0	m13	int
1	alpha237	int
2		

Lex - настраиваемый лексический анализатор

Все лексемы нового языка можно описать (например, КА) и для каждой задать семантику (какую функцию вызывать, если распознана эта лексема – например, обратиться к таблице служебных слов и искать там, если нет, то ...)





Заключение

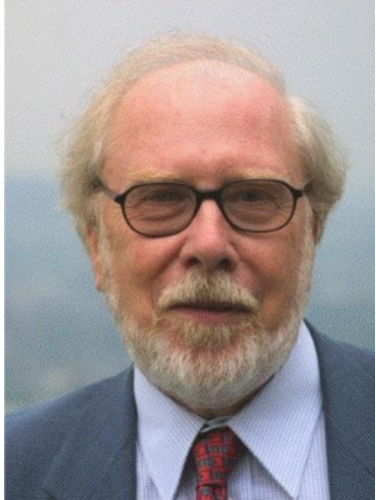
- Модель детерминированного КА может использоваться не только для распознавания, но и для трансляции языков. Для этого на переходах распознающего КА можно указать семантические действия
- Схему распознавателей и трансляторов автоматных языков удобно представлять с помощью синтаксической диаграммы
- Синтаксические диаграммы для сложного автоматного языка удобно строить иерархически, выделяя конструкции языка, и строя синтаксическую диаграмму, задающую каждую конструкцию
- Для трансляции в любое место синтаксической диаграммы может быть помещено семантическое действие
- Для реализации транслятора для каждой конструкции языка по ее синтаксической диаграмме строится своя распознающая процедура. Такая процедура может быть построена по синтаксической диаграмме, которая соответствует детерминированному конечному автомату
- Язык, определяемый набором синтаксических диаграмм, будет автоматным, если в вызовах процедур не будет рекурсии



Заключение (2)

- С помощью этого подхода можно строить трансляторы довольно сложных языков. Большинство скриптовых (интерпретируемых) языков - автоматные
- Входная программа обычно содержит длинные имена, служебные слова, имеет произвольное расположение на листе (произвольное число пробелов и переводов строк), комментарии и т.п.
- Лексемы – это минимальные единицы языка, имеющие смысл. Лексемами являются имена, константы, символы, представляемые группами (например, `+=`, `++`, `:=` и т.д.)
- Лексический анализатор выбрасывает комментарии, 'белые' символы, и т.п., строит лексемы из нескольких символов (например, служебные слова), представляя программу во внутреннем виде как последовательность лексем (терминальных символов грамматики)
- Лексический анализ выполняется с помощью модели конечного автомата, поскольку лексемы описываются автоматом

Персоналии: Никлаус Вирт



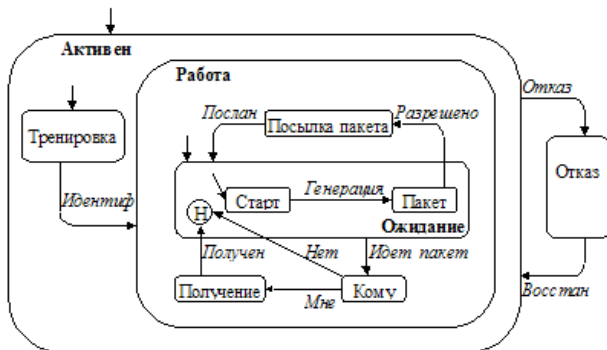
Никлаус Вирт (р.1934) – швейцарский ученый, внесший огромный вклад в теорию и технологию программирования

- Вместе с Энтони Хоаром и Э.Дейкстрой разработал структурное программирование
- Разработал языки Паскаль, Модула, Оберон, Ада, ...
- Разработал пи-код (виртуальной стековой машины) для трансляции
- Придумал синтаксические диаграммы и впервые использовал их для описания языка Паскаль
- Награды:
 - Премия Тьюринга (1984)
 - ACM Award for Outstanding Contributions to Computer Science Education (1987, 1989)
 - ACM Outstanding Research Award in Software Engineering (1999)
 - ...

Один из вариантов курсовой работы

- Построить транслятор с графического языка задания алгоритма обработки данных с помощью графа переходов конечного автомата (либо в виде стейтчартов - карт состояний) в программу на языке Java, либо С, либо Паскаль, ...

Вход: стейтchart
или граф переходов
конечного автомата



Транслятор

Выход: программа,
имитирующая динамику
работы автомата

```
begin
  S := STATE_0; /* установка начального состояния */
  while true do /* бесконечный цикл */
    Next(x); /* в x - очередной входной символ */
    if S = STATE_0 then
      if x=a then y3(); S:= STATE_2; break fi;
      if x=b then y0(); S:= STATE_1; break fi;
      Err0( ) /* ОШ: в сост S0 вход - не а и не b */
    fi;
    if S = STATE_1 then
      if x=a then y3(); S:= STATE_2; break fi;
      if x=b then y1(); S:= STATE_3; break fi;
      Err1( )
    fi;
    ...
    if S = STATE_3 then
      if x=a then y3(); S:= STATE_1; break fi;
      if x=b then y2(); break fi;
      Err3( )
    fi;
  od;
end
```




Спасибо за внимание