

- Дан конечный алфавит  $T$  (**терминальный алфавит**).
  - Примеры:  $T = \{a, b, c, \dots, z\}$ ;  $T = \{0, 1\}$ ;  $T = \{a\}$
- **Цепочка** (строка) – любая конечная последовательность букв из  $T$  (возможно, нулевой длины).
  - Примеры: "a", "aacb", "babababababababa", "0000011111",  $\varepsilon$  (пустая цепочка).
- Множество всех цепочек над  $T$  обозначим  $T^*$ .
- Язык над алфавитом  $T$  – **любое** подмножество цепочек  $L \subseteq T^*$ .
- Все цепочки  $\alpha \in L$  называются **предложениями** этого языка.
- Способы задания языка:
  - перечисление всех предложений (пример: язык Элочки-людоедки);
  - любой алгоритм, принимающий на входе цепочку над  $T$  и возвращающий "да"/"нет" (распознаватель языка);
  - формальная грамматика.

# Формальные грамматики

- **Формальная грамматика** определяется следующей четверкой:
  - **терминальный алфавит**  $T$ ;
  - **нетерминальный алфавит**  $N$  (алфавит для записи правил);
  - **правила вывода цепочек**:  $\alpha_k \rightarrow \beta_k$ ;  $\alpha_k, \beta_k$  – цепочки над  $T \cup N$ ;
  - **начальный символ** (главный нетерминальный символ)  $P$ .
- **Предложениями языка** являются все терминальные цепочки, которые можно получить из  $P$ , конечное число раз применяя правила вывода.
- Н.Хомский (N.Chomsky) в 1957 г. предложил классификацию формальных грамматик: класс 0 – любая грамматика, классы 1, 2 и 3 последовательно сужаются. Практически важны:
  - класс 2: контекстно-свободные (КС) грамматики;
  - класс 3: автоматные (регулярные) грамматики.
- Существуют языки, которые не могут быть описаны грамматикой определенного класса, требуя использования более общего класса. Например:
  - "abcsabcs...abc" ( $N$  раз) – допускает автоматную грамматику;
  - "aaaa...ab...bbbb" (по  $N$  раз каждая буква) – требует КС-грамматики;
  - "aa...ab...bc...cc" (по  $N$  раз каждая) – требует грамматики класса 1.

# Нотация Бэкуса-Наура (расширенная) <sup>3</sup>

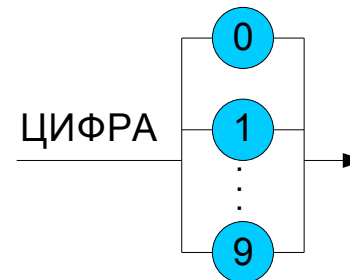
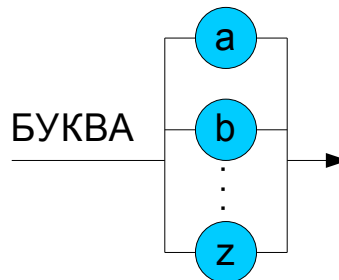
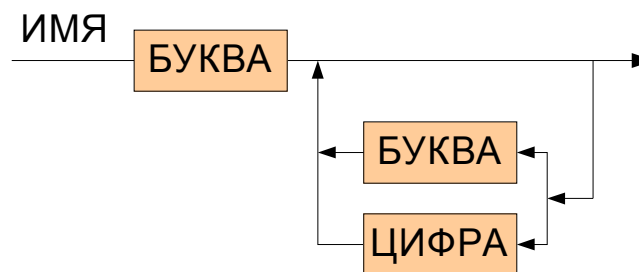
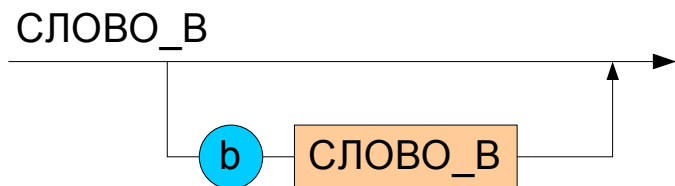
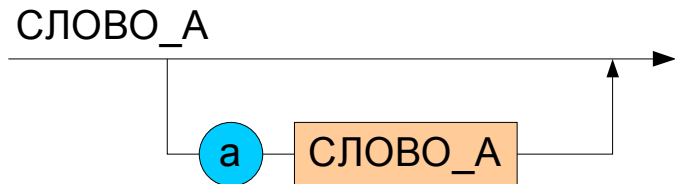
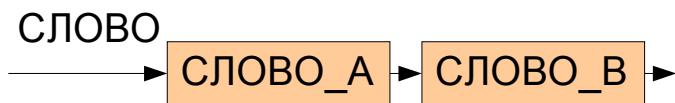
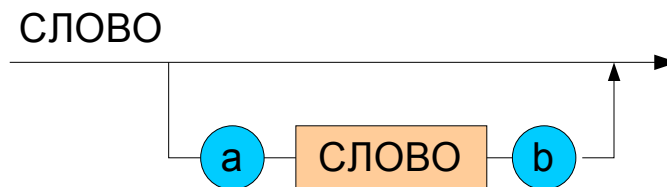
- Классическая БН-нотация: правила вывода записываются в форме:
  - $\langle \text{нетерминал} \rangle ::= \text{цепочка}_1 \mid \text{цепочка}_2 \mid \dots \text{цепочка}_m$
- Нетерминальный алфавит  $N$  состоит из слов-понятий в угловых скобках, а также символов " $::=$ " и " $\mid$ ".
  - " $::=$ " – то же самое, что " $\rightarrow$ ";
  - " $\mid$ " означает выбор одной из нескольких альтернатив, т.е. позволяет объединить запись нескольких правил с одинаковой левой частью.
- Современный расширенный вариант БН-нотации вводит дополнительные обозначения, позволяющие в некоторых случаях резко сократить количество и длину правил:
  - $[ \dots ]$  – обозначает необязательные подцепочки в правой части правила;
  - $\{ \dots \}$  – обозначает повторение подцепочки 0 или более раз.

# Примеры задания языков в БН-нотации

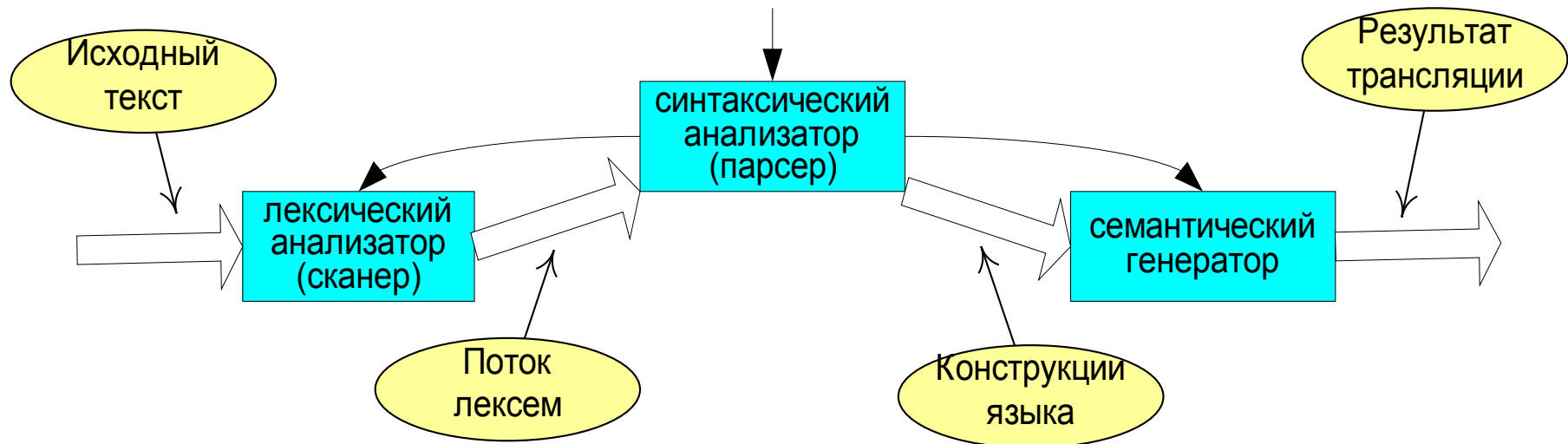
- $\langle \text{СЛОВО} \rangle ::= a \langle \text{СЛОВО} \rangle b \mid \varepsilon$ 
  - Задаёт язык  $L$ , состоящий из всех цепочек вида "aaa...ab...bbb" ( $N$  раз каждая буква,  $N \geq 0$ ).
  
- $\langle \text{СЛОВО} \rangle ::= \langle \text{СЛОВО\_A} \rangle \langle \text{СЛОВО\_B} \rangle$   
 $\langle \text{СЛОВО\_A} \rangle ::= a \langle \text{СЛОВО\_A} \rangle \mid \varepsilon$   
 $\langle \text{СЛОВО\_B} \rangle ::= b \langle \text{СЛОВО\_B} \rangle \mid \varepsilon$ 
  - Задаёт язык вида "aaa...ab...bbb" ( $N$  раз буква 'a',  $M$  раз буква 'b',  $N, M \geq 0$ ).
  
- $\langle \text{ИМЯ} \rangle ::= \langle \text{БУКВА} \rangle \{ \langle \text{БУКВА} \rangle \mid \langle \text{ЦИФРА} \rangle \}$   
 $\langle \text{БУКВА} \rangle ::= a \mid b \mid c \mid d \mid e \mid f \mid \dots \mid x \mid y \mid z$   
 $\langle \text{ЦИФРА} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$ 
  - Задаёт понятно что.
  
- $\langle \text{ОПЕРАТОР\_IF} \rangle ::= \text{if } \langle \text{УСЛОВИЕ} \rangle \text{ then } \langle \text{ОПЕРАТОР} \rangle [\text{else } \langle \text{ОПЕРАТОР} \rangle]$   
 (плюс правила для  $\langle \text{УСЛОВИЕ} \rangle$  и  $\langle \text{ОПЕРАТОР} \rangle$  )
  - Задаёт понятно что.

# Диаграммы Вирта

- Представляют собой наглядную графическую форму, эквивалентную БН-нотации.
- Примеры:



# Структура транслятора



- **Парсер** выполняет синтаксический разбор, т.е. ищет путь вывода от главного нетерминала <ПРОГРАММА> до конкретного текста программы. Распознав очередную конструкцию, вызывает **Генератор** для создания выходного кода.
- **Сканер** (необязательная часть) упрощает работу **Парсера**, разбивая входной текст на лексемы – ключевые слова, имена, числа, операции, разделители.
- **Генератор** либо формирует двоичный код (**компиляция**), либо сразу выполняет команды программы (**интерпретация**).
- Каждый из трех в пределах своей компетенции обнаруживает ошибки.

# Синтаксис, лексика и семантика

- **Синтаксис** языка – грамматическая структура предложения, распознаваемая программой синтаксического анализа (парсером) на основании сопоставления текста с формальной грамматикой языка.
- **Лексика** языка – часть синтаксиса, связанная с разбивкой исходного текста на неделимые языковые единицы-лексемы: ключевые слова, имена переменных, числовые константы, разделители, знаки операций (в том числе составные, как "<=" или ":="). Лексический анализ выполняется на основе регулярной грамматики лексем, облегчая работу парсера.
- **Семантика** языка – содержательный смысл синтаксических конструкций: выполнение операторов, распределение памяти, ведение таблицы имен переменных.
- Практически на семантику сваливают также все ограничения языка, которые трудно или невозможно описать на языке формальных грамматик. Например, использование неописанных переменных считается семантической ошибкой.

# Методы разбора для КС-грамматик

- Существует **алгоритм разбора для КС-грамматики общего вида**, требующий небольшого предварительного «причесывания» грамматики. Общая идея алгоритма: для каждого нетерминала последовательно выясняется, какие подстроки разбираемой строки могут быть из него выведены, а какие нет. (См. пример.)
- **Трудоёмкость**:  $T(N) = O(N^k)$ , где  $k$  зависит от максимального числа нетерминалов в правой части правила вывода. Это может быть приемлемо для некоторых олимпиадных задач, но слишком медленно для трансляции реальных языков программирования, где  $N$  (длина программы) может быть большим. Требуются алгоритмы с линейной оценкой  $O(N)$ .
- Линейные оценки могут быть получены для некоторых подклассов КС-грамматик, таких как  $LL(k)$ -грамматики и  $LR(k)$ -грамматики. Практически любой язык программирования может быть описан грамматикой одного из этих классов, особенно если об этом подумать заранее, при проектировании языка.



# Задача «Словарь»

## ■ Дано:

- алфавит:  $Q$  нетерминалов  $\{A_1, A_2, \dots, A_Q\}$ ,  $R$  терминалов  $\{t_1, t_2, \dots, t_R\}$ ;
- $K$  правил вывода, двух разных типов:
  - $A_i \rightarrow A_j A_k$  (нетерминал  $\rightarrow$  два нетерминала);
  - $A_i \rightarrow t_j$  (нетерминал  $\rightarrow$  терминал);
- начальный символ  $A_1$ ;
- цепочка  $X = "x_1 x_2 \dots x_N"$ .

## ■ Найти: выводима ли цепочка $X$ в данной грамматике.

## ■ Ограничения: $R \leq 26$ , $Q \leq 26$ , $K \leq 20$ , $N \leq 50$ .

# Решение задачи

- Несмотря на простоту структуры грамматики, она может быть достаточно неприятной, поскольку не относится к «хорошим» классам  $LL(1)$ ,  $LR(0)$  и др. Грамматика допускает как левую, так и правую рекурсию.
- Применим общий алгоритм КС-разбора, но приспособленный к данному типу грамматики.
- Пусть  $F(k, i, r) = 1$ , если из нетерминала  $A_k$  выводима подцепочка  $X[i, i+r-1]$ .
- Для подцепочек длины 1 имеем:  $F(k, i, 1) = 1$  тогда и только тогда, когда имеется правило:  $A_k \rightarrow X[i]$ .
- Для подцепочек длины  $> 1$ :  $F(k, i, r) = 1$  тогда и только тогда, когда существует такое правило  $A_k \rightarrow A_p A_q$  и существует разбиение подстроки  $X[i, i+r-1] = X[i, i+a-1] \parallel X[i+a, i+r-1]$  такие, что  $A_p \rightarrow X[i, i+a-1]$  и  $A_q \rightarrow X[i+a, i+r-1]$ .
- Требуется найти  $F(1, 1, N)$ .
- Для расчета можно использовать рекурсию с запоминанием либо рекуррентное заполнение таблицы, начиная с подцепочек длины 1.

# Решение задачи (2)

```
var
  ArrF: array[1..Q, 1..N, 1..N] of -1..1;

function F(k, i, r): Boolean;
  var
    r1, k1, k2: Integer;
begin
  if ArrF[k, i, r] = -1 then begin {Рассчитать значение}
    F := False;
    for r1 := 1 to r-1 do begin
      for k1 := 1 to Q do begin
        if F(k1, i, r1) = 1 then
          for k2 := 1 to Q do
            if F(k2, i+r1, r-r1) = 1 then begin
              ArrF[k, i, r] := 1;
              F := True;
              Exit;
            end;
          end;
        end;
      end;
      ArrF[k, i, r] := 0;
      F := False;
    end
  else {Значение уже рассчитано}
    F := (ArrF[k, i, r] = 1);
  end;
end;
```

# Решение задачи (3)

В основной программе:

Инициализировать массив  $\text{ArrF}[k, i, r] := -1; \{\text{Не рассчитано}\}$

Для всех правил вида  $A[k] \rightarrow X[i]$  присвоить  $\text{ArrF}[k, i, 1] := 1;$

$\text{Answer} := F(A[1], 1, N);$

- **Трудоёмкость:**  $T(N, Q) = O(Q^2 N^3)$

**ВНИМАНИЕ:** алгоритм в бою не проверен!

# LL(1)-грамматики

- Смысл обозначения:
  - входная цепочка просматривается слева (**L**) направо;
  - правые части грамматических правил тоже просматриваются слева (**L**) направо;
  - решение о применимости правила принимается по одной (**1**) текущей букве входной цепочки.
- Самый простой и понятный подкласс LL(1)-грамматик – **S-грамматики**, которые удовлетворяют таким условиям:
  - правые части всех правил начинаются с терминального символа;
  - для одного и того же нетерминала все правые части начинаются с разных терминалов.
- В случае LL(1)-грамматики общего вида для каждого правила с нетерминалом  $A$  в левой части определяется **множество выбора**, которое включает в себя:
  - все терминалы, которые могут быть первыми буквами цепочек, выводимых из  $A$ ;
  - если правило имеет вид  $A \rightarrow V\alpha$  и из  $V$  выводима пустая цепочка, то все терминалы, которые могут быть первыми буквами цепочек, выводимых из  $\alpha$ .
- Грамматика относится к классу LL(1), если разные правила для одного и того же нетерминала имеют непересекающиеся множества выбора.
- Для грамматик LL(1) работает алгоритм рекурсивного спуска, который выполняет построение вывода от начального символа до терминальной цепочки, просматривая эту цепочку один раз слева направо.

# Алгоритм рекурсивного спуска для LL(1)-грамматики

- Для каждого нетерминала пишется процедура, которая по очередной входной букве выбирает правую часть правила, проверяет соответствие входных терминалов правилу и вызывает (возможно, рекурсивно) процедуры для нетерминалов в правой части правила.
- Весь разбор инициализируется вызовом процедуры для главного нетерминала.
- Написание процедур выполняется достаточно формальным образом на основании формы Бэкуса-Наура или (еще лучше) диаграмм Вирта.
- Если требуется семантическая обработка (например, подсчет выражения или генерация кода), то соответствующие операторы или вызова семантических процедур вставляются (к сожалению, неформально) в подходящие места процедур разбора нетерминалов.
- **Трудоемкость** разбора:  $T(N) = O(N)$ .

# Задача «Химические реакции»

- **Дано:** правила записи химических реакций:
  - ❑  $\langle \text{уравнение} \rangle ::= \langle \text{формула} \rangle "=" \langle \text{формула} \rangle$
  - ❑  $\langle \text{формула} \rangle ::= [\langle \text{число} \rangle] \langle \text{молекула} \rangle \{ "+" [\langle \text{число} \rangle] \langle \text{молекула} \rangle \}$
  - ❑  $\langle \text{молекула} \rangle ::= \langle \text{элемент} \rangle [\langle \text{число} \rangle] \{ \langle \text{элемент} \rangle [\langle \text{число} \rangle] \}$
  - ❑  $\langle \text{элемент} \rangle ::= \langle \text{прописная} \rangle [\langle \text{строчная} \rangle]$
  - ❑  $\langle \text{прописная} \rangle ::= "A" \dots "Z"$
  - ❑  $\langle \text{строчная} \rangle ::= "a" \dots "z"$
  - ❑  $\langle \text{число} \rangle ::= "2" \dots "9"$
- **Требуется:** определить правильность записи реакции, включая проверку равенства числа атомов справа и слева.
- **Примеры:**
  - ❑  $\text{H}_2\text{SO}_4 + 2\text{NaOH} = \text{Na}_2\text{SO}_4 + 2\text{H}_2\text{O}$  – правильно
  - ❑  $\text{H}_2\text{SO}_4 + 2\text{NaOH} = \text{Na}_2\text{SO}_4 - 2\text{H}_2\text{O}$  – нарушение синтаксиса
  - ❑  $2\text{PbO} + 3\text{Xx} = 5\text{Au}$  – алхимическое превращение, семантическая ошибка

# Решение задачи

- Для каждого нетерминала определим множество **First** – множество допустимых начальных букв терминального алфавита.
  - $\text{First}(\text{число}) = "23456789"; \quad \text{First}(\text{строчная}) = "abcde...xyz";$
  - $\text{First}(\text{прописная}) = "ABCDE...XYZ";$
  - $\text{First}(\text{элемент}) = \text{First}(\text{молекула}) = "ABCDE...XYZ";$
  - $\text{First}(\text{формула}) = \text{First}(\text{уравнение}) = "23456789ABCDE...XYZ";$
- Глобальная переменная **curChar** – текущая буква входного текста.
- Глобальная переменная **curNumb** – текущее число, найденное в тексте.
- Процедура **GetNext**: читает следующую букву в переменную curChar.
- Процедура **Check**("символы") – проверяет принадлежность curChar к заданному множеству букв, допустимых в данной позиции. В противном случае прекращает разбор и выдает ответ "нет".
- Для каждого нетерминала пишется процедура разбора. Предполагается, что в момент вызова curChar = первой букве разбираемого нетерминала, а после выхода из процедуры curChar = первой букве, идущей после распознанного нетерминала.
- **Синим шрифтом** в процедурах выделены семантические действия (все остальное – чистый синтаксис).



# Решение задачи (2)

```

procedure PEquation;
begin
    Начать счет атомов слева;
    Check(First(уравнение));
    PFormula;
    Check("=");
    GetNext;
    Начать счет атомов справа;
    PFormula;
    Сравнить число атомов справа и
    слева, выдать "да" или "нет".;
end;

procedure PFormula;
begin
    Check(First(формула));
    repeat
        numMolec := 1;
        if curChar ∈ First(число)
        then begin
            PNumber;
            numMolec := curNumb;
        end;
        PMolecula;
        flag := (curChar = "+" );
        if flag then
            GetNext;

```

```

        until not Flag;
    end;

    procedure PMolecula;
    begin
        Check(First(молекула));
        repeat
            PElement;
            numElem := 1;
            if curChar ∈ First(число) then
                begin
                    PNumber;
                    numElem := curNumb;
                end;
            Увеличить число атомов данного
            элемента на numMolec * numElem;
        until curChar ∉ First(элемент);
    end;

    procedure PElement;
    begin
        Check(First(элемент));
        PUpCase;
        if curChar ∈ First(строчная) then
            PLoCase;
        Найти или добавить в таблицу имя
        текущего элемента;
    end;

```

# Решение задачи (3)

```
procedure PUpCase;  
begin  
    Check(First(прописная));  
    Запомнить первую букву имени;  
    GetNext;  
end;  
  
procedure PLoCase;  
begin  
    Check(First(строчная));  
    Запомнить вторую букву имени;  
    GetNext;  
end;  
  
procedure PNumber;  
begin  
    Check(First(число));  
    curNumb := значение цифры;  
    GetNext;  
end;
```

В основной программе:

```
GetNext;  
PEquation;
```