



# 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](mailto:karpov@dcn.infos.ru)



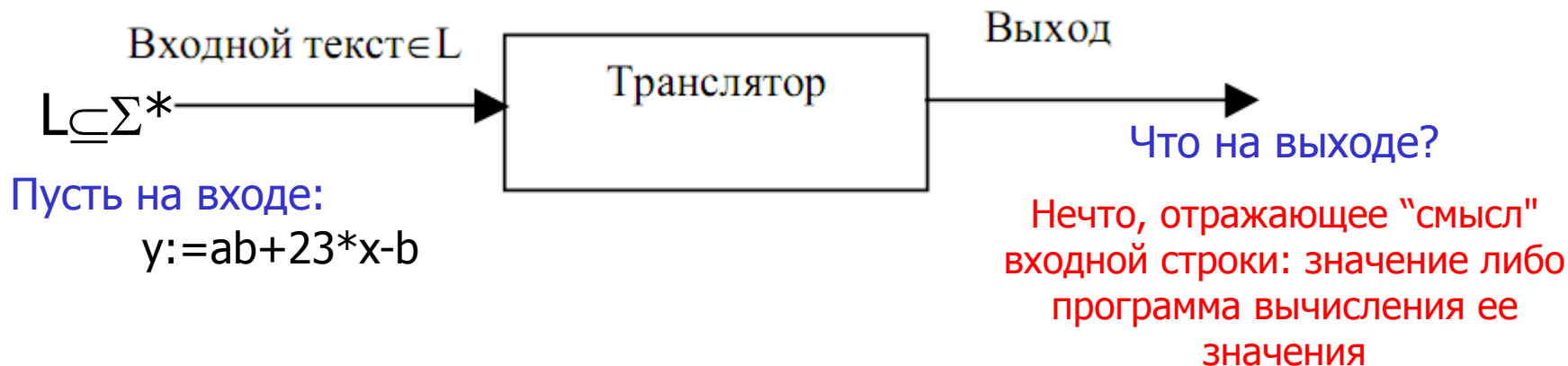
# Структура курса

---

- Конечные автоматы-распознаватели – 4 л
- Порождающие грамматики Хомского – 3 л
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
  - Лекция 8. Атрибутная семантика Кнута и атрибутные грамматики
  - Лекция 9. Примеры атрибутных трансляций. Трансляция арифметических выражений
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции - 2 л

## Как выявить смысл в цепочке языка?

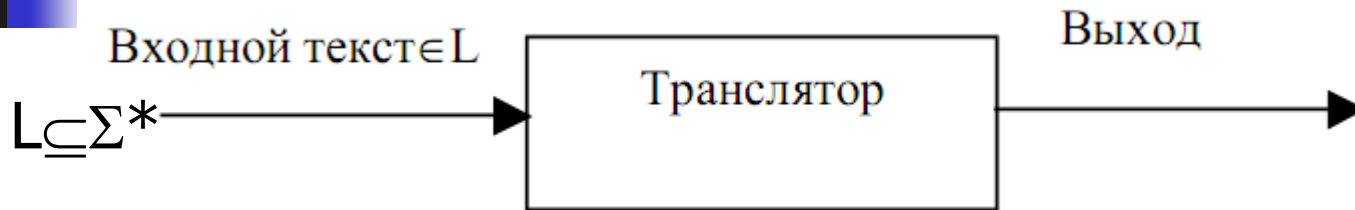
Выход транслятора должен отражать смысл входной строки



- С каждой строкой и естественного, и формального языка связан смысл (значение, семантика) этой строки.
- Например, если строка  $X+Y$  – это "правильная" цепочка языка арифметических выражений, где  $X$  и  $Y$  – тоже выражения, а '+' – символ сложения, то под "значением" цепочки  $X+Y$  естественно понимать сумму значений выражения  $X$  и выражения  $Y$

Как выявить смысл в цепочке языка?

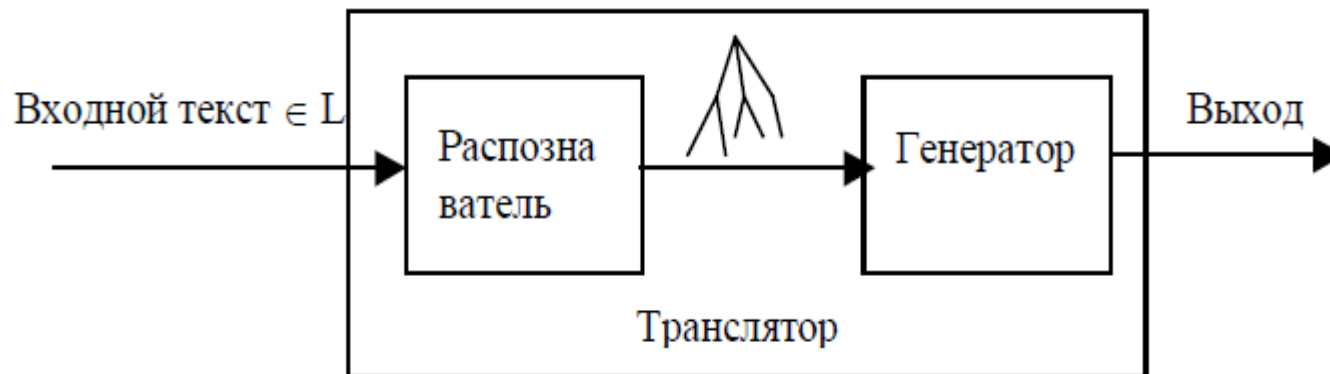
# Как строить транслятор??



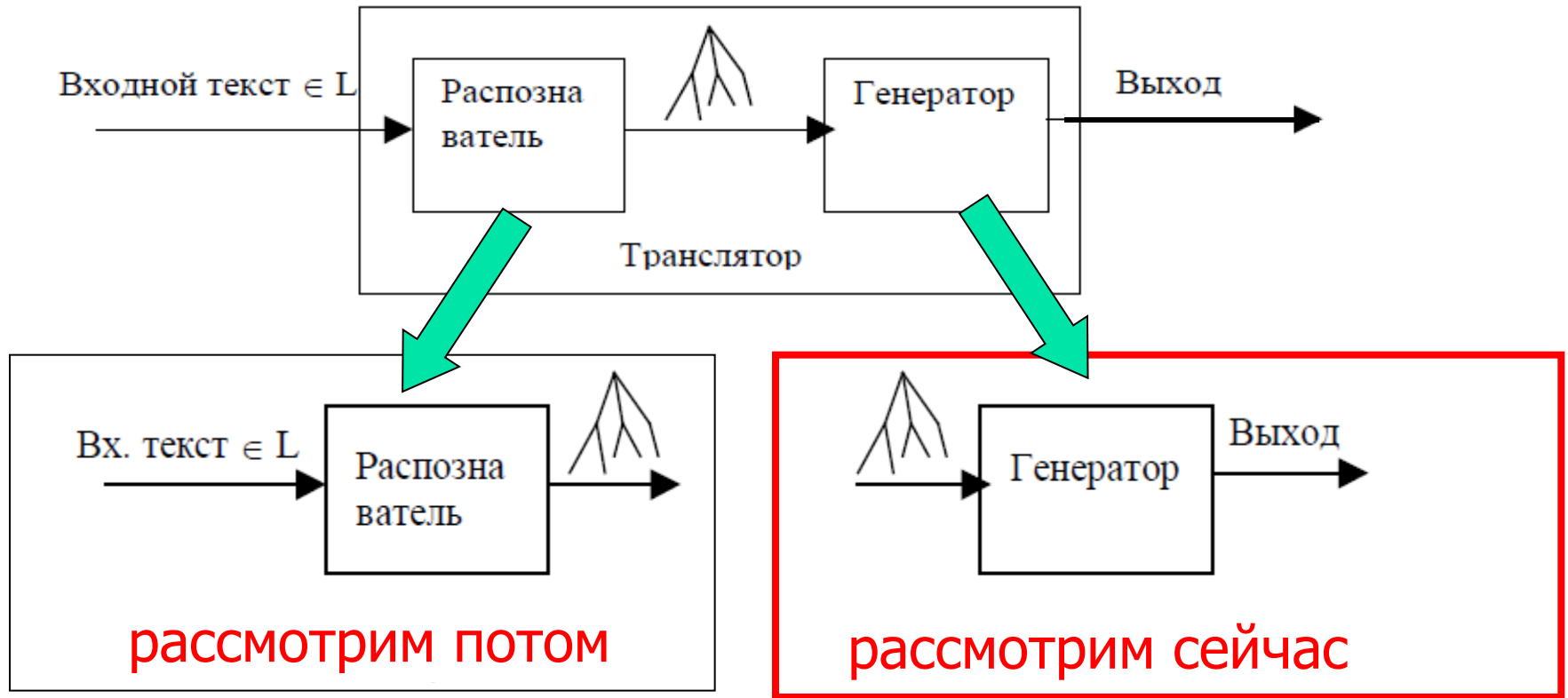
Идея Хомского: структура входной цепочки помогает пониманию



Транслятор строится из ДВУХ блоков: Распознавателя и Генератора



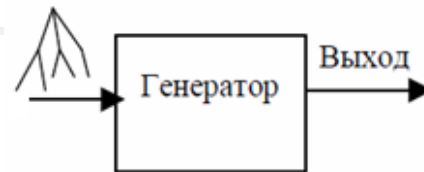
## Фазы трансляции



Будем считать, что дерево вывода (структура цепочки) уже построено

Как структура цепочки помогает восстановить тот смысл, который несет цепочка?

# Определения **смысла** предложений



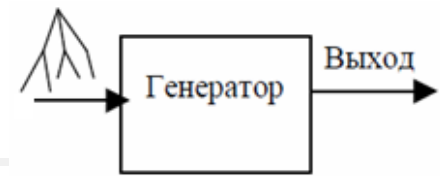
- ❑ Не существует правил, позволяющих определить смысл слова по его звучанию, смысла по начертанию символа, смысла знака по его изображению
- ❑ Связь звучания слова, начертания символа с их смыслом должна быть заучена. Словарь любого языка конечен, и такое соответствие всегда определено

Чтобы пользоваться языком, нужно выучить значения всех слов словаря

- Как вычислить смысл предложения, зная смыслы слов (символов языка)?
- Язык состоит из бесконечного множества предложений, поэтому сопоставление каждого предложения его смыслу не может быть задано таблицей или заучено: это потребовало бы бесконечных ресурсов.
- В соответствии с гипотезой Хомского, семантика, смысл входной цепочки языка **вычисляется** на основе дерева вывода в порождающей ее грамматике

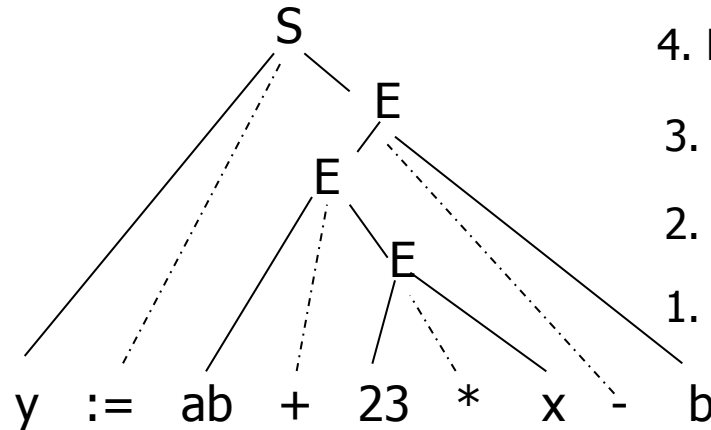
## Но КАК???

# Структура цепочки КС-языка



$y := ab + 23 * x - b$

Как понимать?



4. Выполняем присваивание

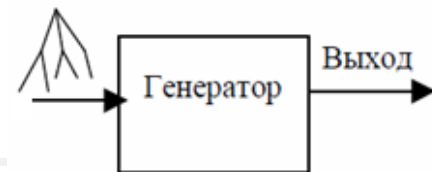
3. Выполняем вычитание

2. Выполняем сложение

1. Выполняем умножение

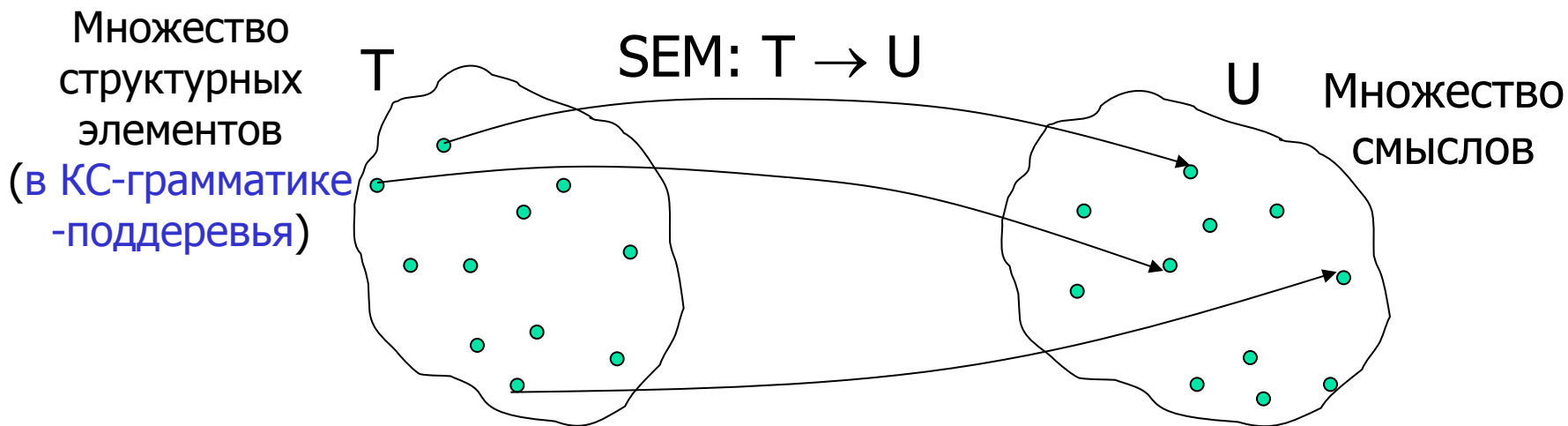
- Именно дерево вывода цепочки языка в КС-грамматике и является той искомой структурой цепочки, с которой мы связывали надежды на простоту алгоритма вычисления "смысла" при построении алгоритмов трансляции языков. Это предположение и сделал Н.Хомский.
- Нетерминалы грамматики представляют собой языковые конструкции (синтаксические классы), играющие определенные роли в предложениях языка, а правила грамматики показывают связи этих конструкций и их подконструкций, которые эти роли, фактически, исполняют
- Дерево вывода показывает, как эти связи работают в данной входной цепочке
- На основании таких связей можно попытаться построить, "вычислить" смыслы более общих конструкций из смыслов их составляющих конструкций и, в конце концов, смысл всего предложения

# Что такое семантика языка?



“Смысл мира должен лежать вне его” (Людвиг Витгенштейн)

Семантика - отображение из множества структурных элементов языка (деревьев и поддеревьев вывода) в множество смыслов



- Вопросы семантики очень сложны. Существуют довольно сложные теории формальной семантики естественных языков
- Смыслы, которые можно приписать символам, фразам и предложениям языка – это в общем случае некоторое множество элементов любой природы

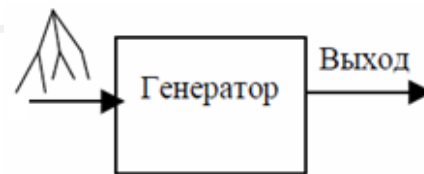


# Атрибутная семантика – связь структуры и значения

В семантике желательно использовать принцип композициональности:

*значение, приписываемое всей структуре, должно строиться из значений, приписываемых ее частям, на основе связей между этими частями*

Принцип композициональности в лингвистике: значение составного выражения - это функция значений его частей и способов их соединения в синтаксическую структуру

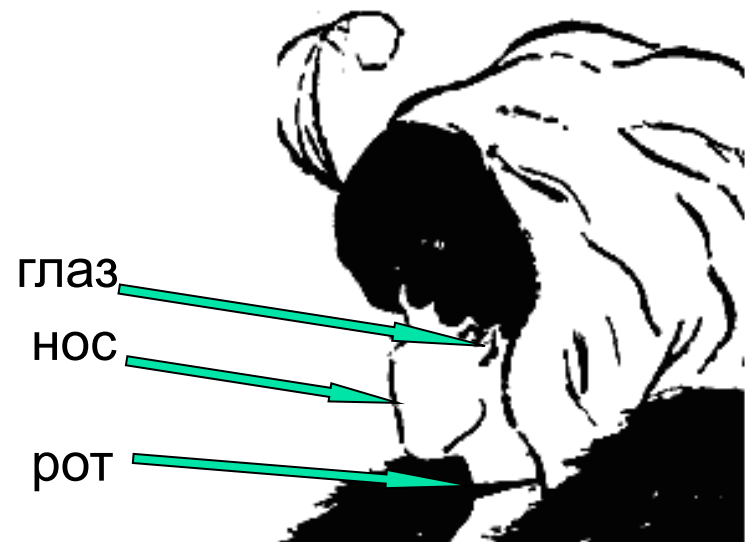


**В области трансляции эта идея представлена Дональдом Кнутом: атрибутная семантика**

- **АТРИБУТНАЯ СЕМАНТИКА** – это один из подходов к тому, каким образом структура цепочки (дерево вывода в порождающей КС-грамматике), может быть использована для генерации требуемого выхода транслятора

# Пример неоднозначного образа: милая дама или злобная теща???

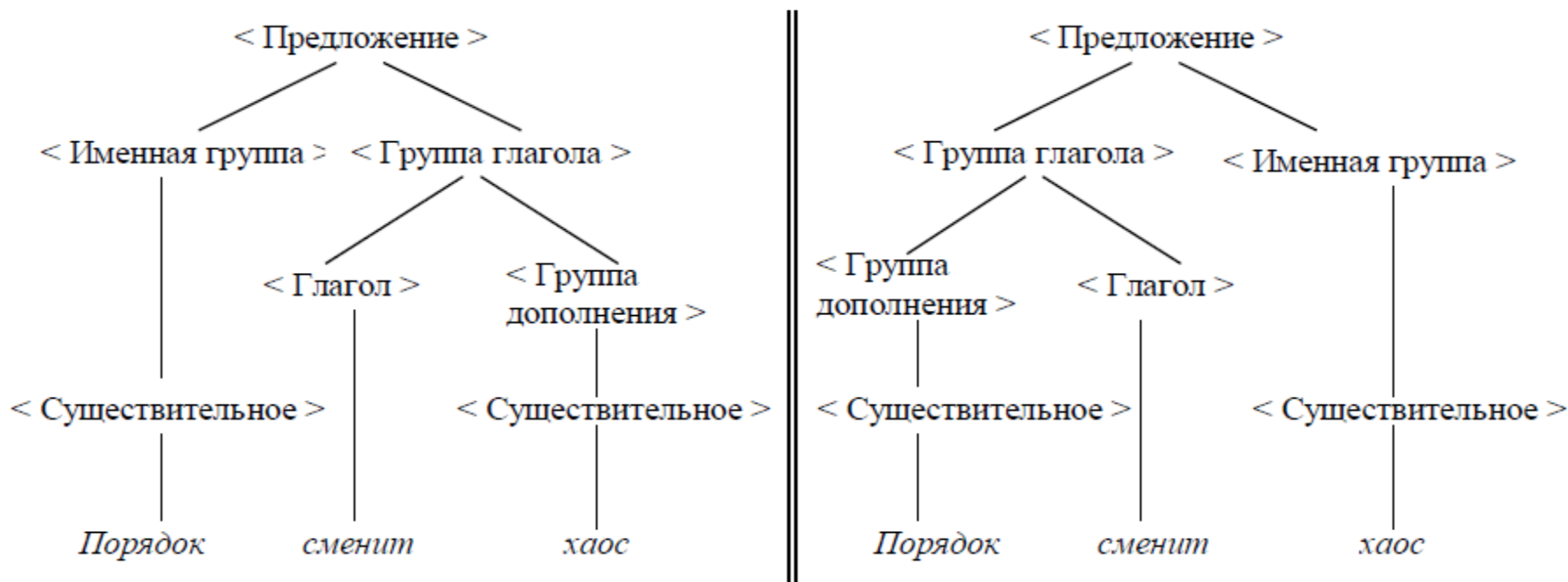
Для фиксации одного смысла нужно связать элементы рисунка (символы словаря!) с абстрактной "конструкцией": нос, ухо, подбородок, ...



Пока такой связи не установлено, трактовать всю картинку однозначно нельзя

# Различные структуры двусмысленного предложения

Идея Хомского: понимание смысла предложения – по его грамматической структуре, определяющей роль каждого слова и отдельных групп в предложении

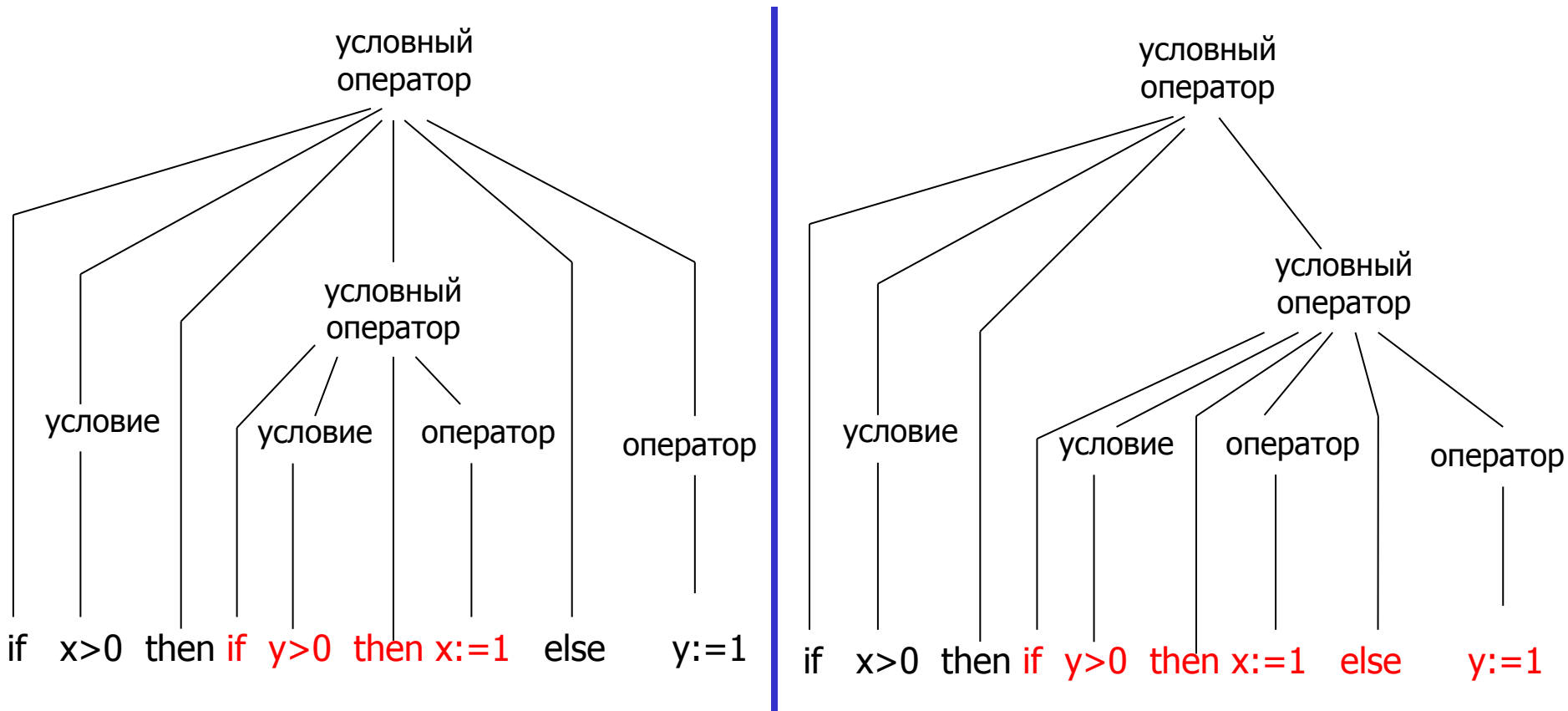


Можно приписать словам *порядок* и *хаос* разные роли – либо роль подлежащего (активного объекта, именной группы), либо дополнения (объекта, на который направлено действие). Структура предложения может пониматься двояко  $\Rightarrow$  у предложения два смысла

<Именная группа>, <Группа дополнения> - НАЗВАНИЯ КОНСТРУКЦИЙ, НЕТЕРМИНАЛЫ

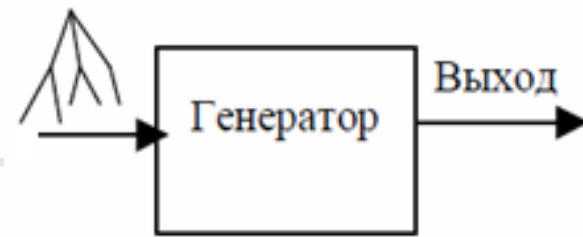
# Как оттранслировать условный оператор?

if  $x > 0$  then if  $y > 0$  then  $x := 1$  else  $y := 1$



Сгенерированные команды будут разными при разных структурах

# Основной вопрос трансляции

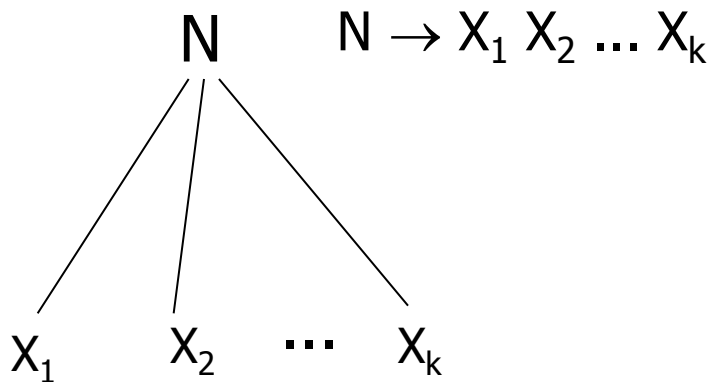


## Смысл цепочки – по дереву вывода!

- Как сопоставить каждому дереву вывода цепочки смысл, семантику этой цепочки? Деревьев столько же, сколько цепочек языка (и, возможно, больше, если грамматика двусмысленна)
- Деревьев вывода тоже бесконечное количество, как и входных цепочек!

Деревьев вывода бесконечное число, но типов структур – правил грамматики – конечное число

*Для арифметического выражения это сумма двух подвыражений ( $E+E$ ), их произведение ( $E * E$ ) и т.д. (они задаются грамматикой и есть в дереве!)*

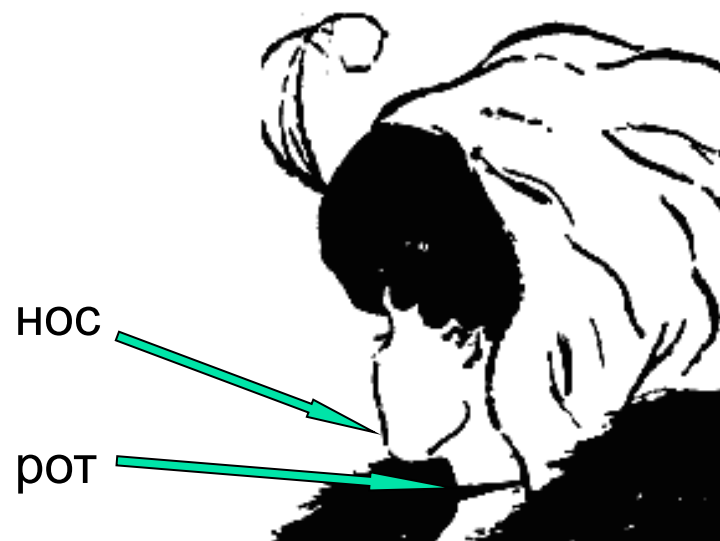
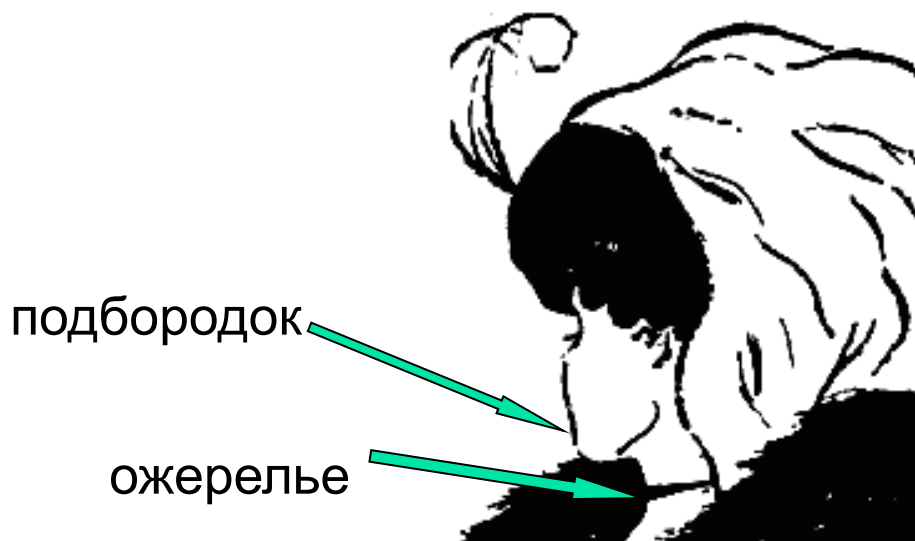


Все типы структур (узлов дерева) задаются правилами грамматики

Смысл удобно связать с ПРАВИЛАМИ грамматик

# Структура и значение

- Как формально (алгоритмически) выразить тот факт, что смысл цепочки языка можно определить по его структуре (дереву вывода)?



- Введем смысловые параметры (семантические атрибуты) для каждого терминального символа (символа языка) и каждого нетерминального символа (конструкции языка)
- В правилах грамматики (правилах построения конструкций из подконструкций и терминалов) определим связи между семантическими атрибутами

## С правилами грамматики связываются зависимости между семантическими атрибутами

- Введем смысловые параметры (семантические атрибуты) для каждого терминального символа (символа языка) и каждого нетерминального символа (конструкции языка)

1.  $E \rightarrow E' + E''$

2.  $E \rightarrow i$

$E.v := E'.v + E''.v$

$E.v := i.v$

символ в цепочке языка

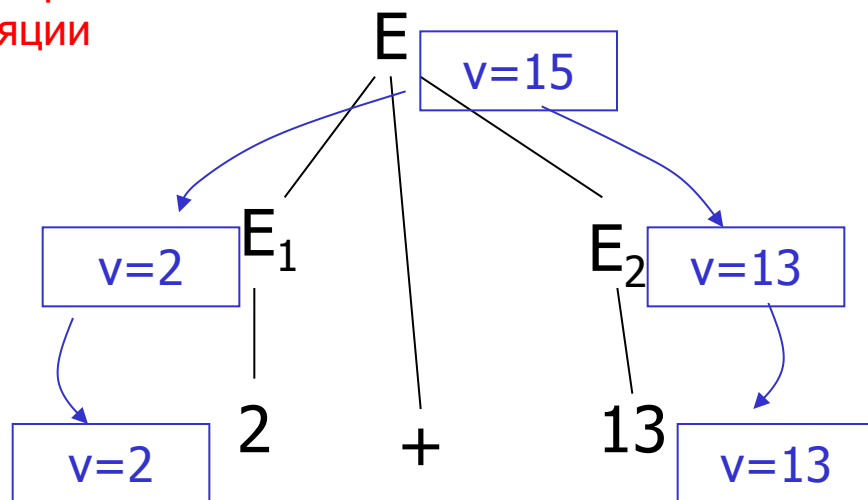
операция над  
значениями при  
выполнении трансляции

$E.v$  – атрибут  $v$  нетерминала  $E$

$i.v$  – атрибут  $v$  терминала  $i$

Значения атрибутов  
вычисляются по дереву

Свяжем с каждой нетерминальной конструкцией  $E$ , имеющей смысл "выражение", семантический параметр (атрибут)  $E.v$  – (value) – значение того выражения, которое выводится из этой конструкции  $E$

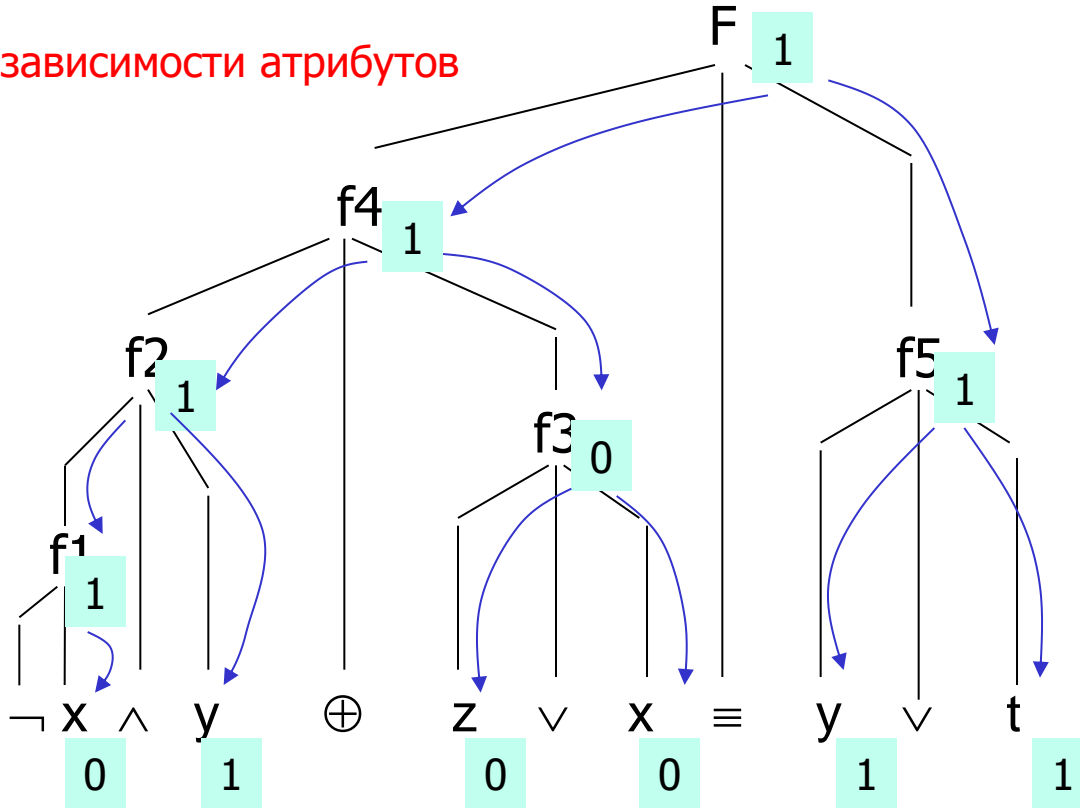


зависимости атрибутов

# Вычисление значения логической функции по синтаксическому дереву формулы

$\neg x \wedge y \oplus z \vee x \equiv y \vee t$  - Как понимать?

зависимости атрибутов



$f1.v = \neg x.v$   
 $f2.v = f1.v \wedge y.v$   
 $f3.v = z.v \vee x.v$   
 $f4.v = f2.v \oplus f3.v$   
 $f5.v = y.v \vee t.v$   
 $F.v = f4.v \equiv f5.v$

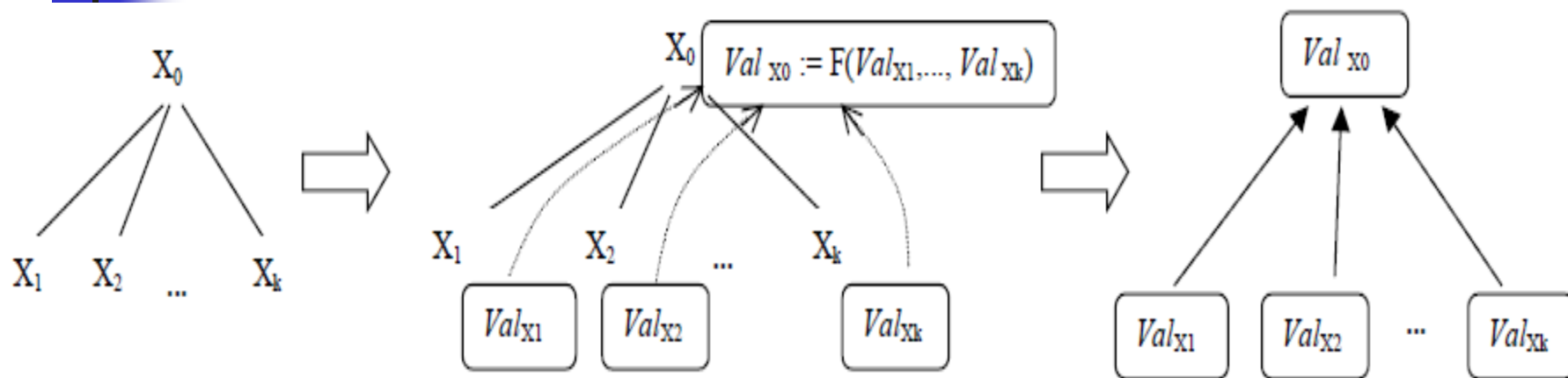
Приоритеты:

$\neg$  НЕ  
 $\wedge$  И  
 $\vee$  ИЛИ  
 $\Rightarrow, \oplus, \equiv, \dots$  остальные

$x.v=0; y.v=1; z.v=0; t.v=1$



## Атрибутная грамматика: вычисление значений атрибутов



Узел дерева вывода

Замещающая  
семантическая структура

Вычисление синтезируемых  
атрибутов

В общем случае каждому нетерминальному символу может быть сопоставлено несколько атрибутов произвольных типов. С каждой продукцией контекстно-свободной грамматики связываются правила вычисления атрибутов входящих в продукцию символов (семантические правила)

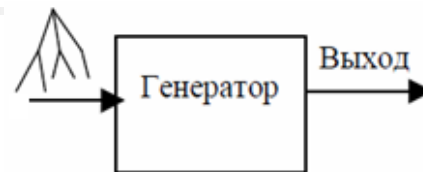
Терминалы – это лексемы, их семантическими атрибутами обычно являются значения, вычисленные при лексическом анализе и хранящиеся в таблице СИМВОЛОВ

# Локальные зависимости семантических атрибутов

Хотя арифметических выражений бесконечно много, но типов структур, которые могут комбинироваться в выражениях, конечное число – это разность двух подвыражений ( $E - E$ ), их произведение ( $E * E$ ) и т.д. (и все они задаются правилами грамматики!)

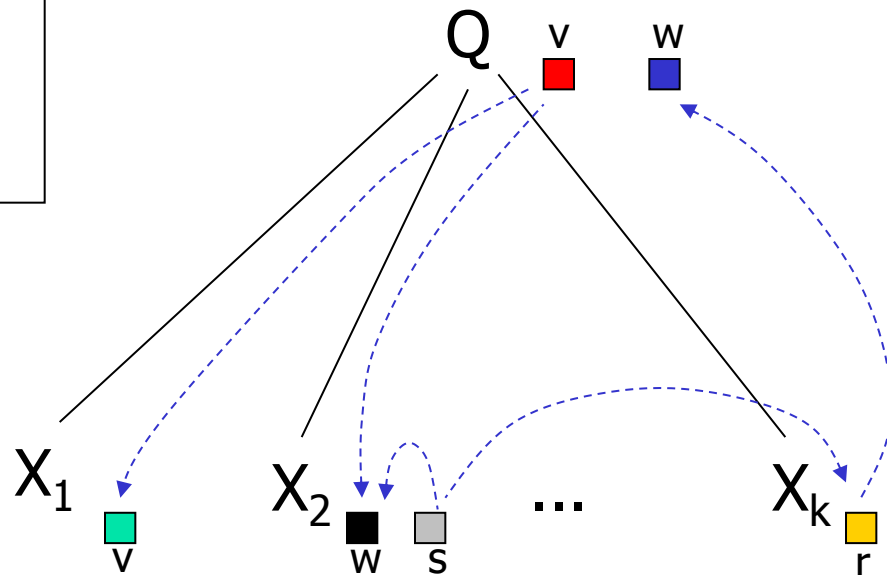
$E \rightarrow E' - E''$	$E.v := E'.v - E''.v$
$E \rightarrow E' * E''$	$E.v := E'.v \times E''.v$
$E \rightarrow i$	$E.v := i.v$

- Все связи между семантическими атрибутами определяются ЛОКАЛЬНО внутри правил грамматики (правил построения конструкций из подконструкций и терминалов)



$$Q \rightarrow X_1 X_2 \dots X_k \in R$$

$Q.v := f_1(X_1.v, X_2.s)$
$X_2.s := f_2(X_2.w, X_k.r)$
$X_k.r := f_3(Q.w)$



зависимости семантических атрибутов

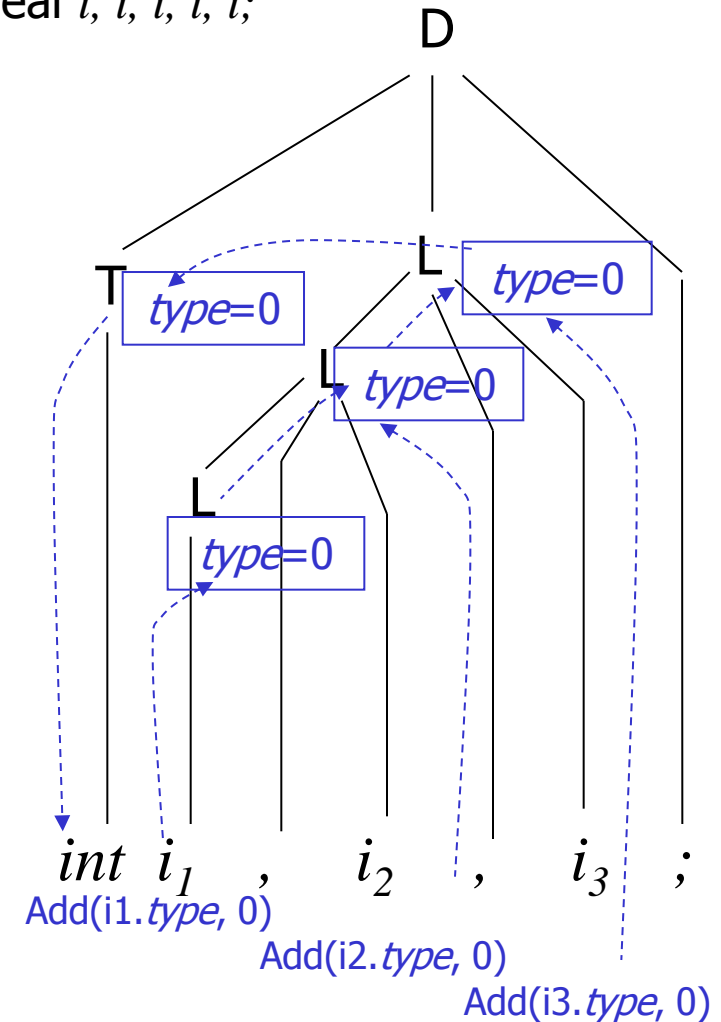
# Синтезируемые и наследуемые атрибуты

- Кнут: два типа семантических атрибутов: синтезируемые и наследуемые
  - **Синтезируемые атрибуты** нетерминала определяются через атрибуты его потомков. В дереве вывода они вычисляются снизу вверх. Пример – значение выражения, значения известны для листьев дерева (операндов), а для родителей (подвыражений) вычисляется по значениям их детей
  - **Наследуемые атрибуты** нетерминала определяются через семантические атрибуты его непосредственного предка. В дереве вывода они вычисляются сверху вниз. Пример - таблица символов. Она строится для корневого элемента, затем передается вниз к листьям, наполняясь по пути новыми объявлениями имен. Другие примеры: вес цифры в дробной части числа зависит от ее положения, тип переменной зависит не от того, какое у нее имя, а от того, в какую конструкцию описания (**integer** либо **real**) входит ее имя
- Если говорить в терминах продукций (правил грамматики), то синтезируемые атрибуты – это такие атрибуты нетерминала в левой части продукции, которые определяются через атрибуты символов правой части. Наследуемые атрибуты – это те атрибуты символов правой части продукции, которые определяются через атрибуты любых других символов этой же продукции
- Разницу между этими двумя типами атрибутов можно понять так:
  - синтезируемые атрибуты конструкции языка (нетерминала грамматики) зависят от того, из чего строится эта конструкция
  - наследуемые атрибуты синтаксической конструкции зависят от того, в какую другую конструкцию и каким образом входит данная конструкция

# Пример: атрибутная грамматика описаний типов

Примеры порождаемых цепочек: `int i, i, i, i; real i, i, i, i, i;`

N	Продукции	Семантические правила
1.	$D \rightarrow T L;$	$type(L) := type(T)$
2.	$T \rightarrow \text{int}$	$type(T) := 0$
3.	$T \rightarrow \text{real}$	$type(T) := 1$
4.	$L \rightarrow L_1, i$	$type(L_1) := type(L)$ $Add(i.type, type(L))$
5.	$L \rightarrow i$	$Add(i.type, type(L))$



$Add(i.type, t)$  помещает  $t$  в поле `type` переменной с именем  $i$  таблицы имен

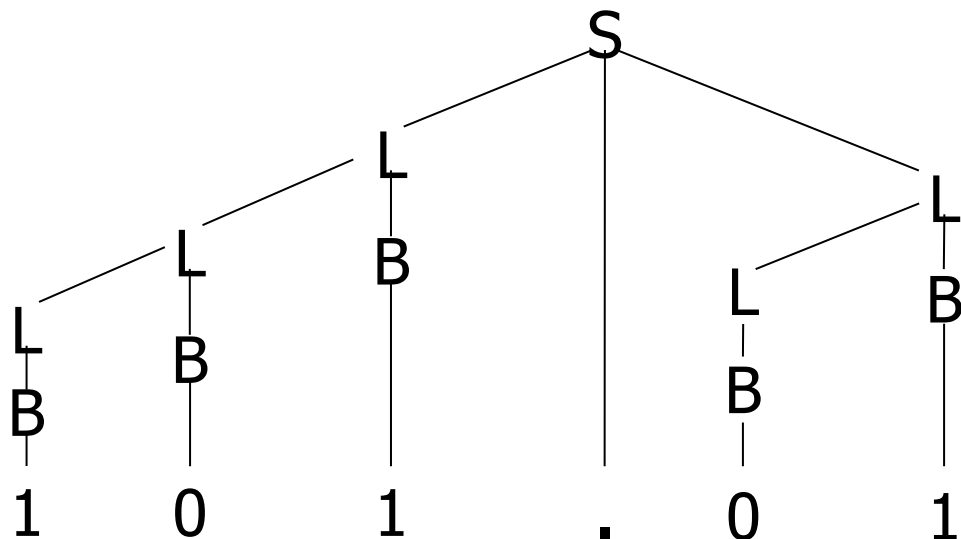
## Пример: формальное определение двоичных чисел

- Как определить двоичные числа? Их бесконечное число!

Каждому числу **грамматика** сопоставляет дерево вывода:

КС-грамматика!

G::	1	$S \rightarrow L . L$
	2	$L \rightarrow L B$
	3	$L \rightarrow B$
	4	$B \rightarrow 0$
	5	$B \rightarrow 1$



Грамматика – это конечное число правил, определяющих, как строить **любое** двоичное число из 0, 1 и точки

Но как подсчитать **значение** любого построенного двоичного числа?

Хотя двоичных чисел может быть бесконечно много, мы хотим построить конечное число **семантических** правил. Семантические правила определяем так, чтобы значение всей записи строилось из значений составляющих ее частей



## Пример (продолжение)

Определим семантику двоичной цепочки как ее численное значение. В частности, по цепочке 110.01 компилятор должен выдать величину 6.25 (в машинном представлении). Припишем каждому нетерминалу семантические атрибуты следующим образом:

Нетерминал	Атрибут	Примечание
B	$v(B)$	$v(B)$ – целочисленное значение, приписываемое двоичному символу, представленному B
L	$v(L)$	$v(L)$ – целочисленное значение, приписываемое цепочке двоичных символов, выведенных из L
	$l(L)$	$l(L)$ – длина цепочки двоичных символов, выведенных из L
S	$v(S)$	$v(S)$ – численное значение, приписываемое двоичной цепочке символов, выведенных из S

Семантические атрибуты и их зависимости разработчик выбирает САМ

## Пример (продолжение)

Построим атрибутную грамматику этого языка:

N	Продукции	Семантические правила
1.	$S \rightarrow L_1.L_2$	$v(S) := v(L_1) + v(L_2)/2^{l(L_2)}$
2.	$L_0 \rightarrow L_1B$	$v(L_0) := 2 v(L_1) + v(B); \quad l(L_0) := l(L_1) + 1$
3.	$L \rightarrow B$	$v(L) := v(B); \quad l(L) := 1$
4.	$B \rightarrow 0$	$v(B) := 0;$
5.	$B \rightarrow 1$	$v(B) := 1;$

Здесь значения атрибутов всех нетерминалов определяются через значения атрибутов их непосредственных потомков в правых частях правил грамматики. Теперь структуру дерева вывода можно расширить, добавив атрибуты и их значения во всех узлах.

# Формальное определение 'значения' двоичных чисел

G::

- 1  $S \rightarrow L . L$
- 2  $L \rightarrow L B$
- 3  $L \rightarrow B$
- 4  $B \rightarrow 0$
- 5  $B \rightarrow 1$

Атрибуты:

B: B.v

L: L.v, L.l

S: S.v

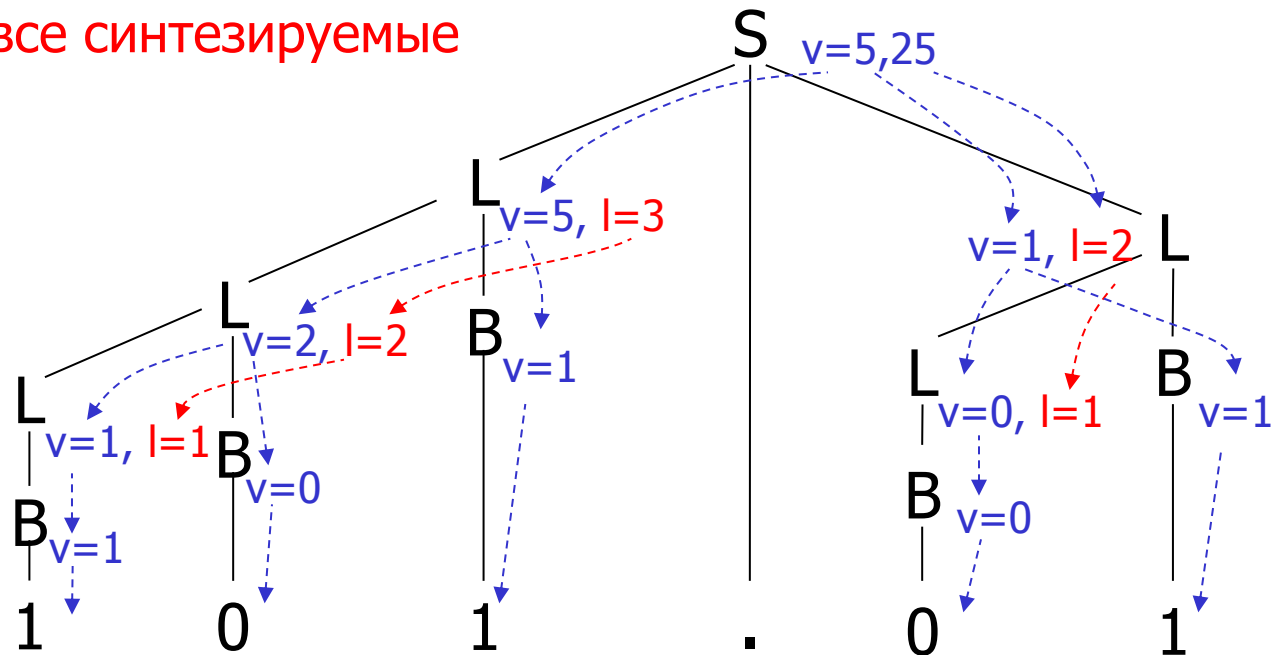
L.v – 'значение' L

L.l – 'длина' L

Синтезируемые атрибуты определяются через атрибуты потомков

n	Синтаксическое правило	Семантическое правило
1	$S \rightarrow L' . L''$	$S.v := L'.v + L''.v/2$ <span style="color: red;">L'.l</span>
2	$L' \rightarrow L'' B$	$L'.v := L''.v * 2 + B.v$ ; $L'.\textcolor{red}{l} := L''.\textcolor{red}{l} + 1$
3	$L \rightarrow B$	$L.v := B.v$ ; $L.\textcolor{red}{l} := 1$ ;
4	$B \rightarrow 0$	$B.v := 0$
5	$B \rightarrow 1$	$B.v := 1$

все синтезируемые





## Другая семантика грамматики двоичных чисел

Припишем нетерминалам грамматики следующие атрибуты:

Нетерминал	Атрибут	Примечание
B	$v(B)$	$v(B)$ – целочисленное значение, приписываемое двоичному символу, представленному B
	$s(B)$	$s(B)$ – масштаб (вес) данного двоичного символа B
L	$v(L)$	$v(L)$ – значение, приписываемое подцепочке двоичных символов, выведенных из L, с учетом их положения во всей цепочке
	$l(L)$	$l(L)$ – длина цепочки двоичных символов, выведенных из L
	$s(L)$	$s(L)$ – масштаб (вес) цепочки двоичных символов, выведенных из L
S	$v(S)$	$v(S)$ – значение, приписываемое двоичной цепочке символов, выведенных из S

На основе этой таблицы атрибутов можно построить другую семантику атрибутивной грамматики двоичных цепочек

## Атрибутная грамматика двоичных рациональных чисел

N	Продукции	Семантические правила
1.	$S \rightarrow L_1.L_2$	$v(S) := v(L_1) + v(L_2); \quad s(L_1) := 0; \quad s(L_2) := -l(L_2)$
2.	$L_0 \rightarrow L_1 B$	$v(L_0) := v(L_1) + v(B); \quad s(B) := s(L_0); \quad s(L_1) := s(L_0) + 1; \quad l(L_0) := l(L_1) + 1$
3.	$L \rightarrow B$	$v(L) := v(B); \quad s(B) := s(L); \quad l(L) := 1$
4.	$B \rightarrow 0$	$v(B) := 0 \times 2^{s(B)} = 0;$
5.	$B \rightarrow 1$	$v(B) := 1 \times 2^{s(B)} = 2^{s(B)}$

Каждый бит, порождаемый из нетерминала  $B$ , имеет свой индивидуальный вес  $2^{s(B)}$  в цепочке битов, с которым он входит в число. Этот вес определяется масштабом  $s$

Масштаб  $s$  указывает положение этого символа относительно запятой

Атрибут  $s$  приписан и цепочке символов  $L$ , он имеет значение положения относительно запятой крайнего правого бита цепочки, выводимой из  $L$

В общем случае из-за наследуемых атрибутов последовательность вычислений по аннотированному дереву вывода может проходить дерево не только снизу вверх (как для синтезируемых), но и в различных направлениях

# Формальное определение 'значения' двоичных чисел

n	Синтаксические правила	Семантические правила
1	$S \rightarrow L' . L''$	$N.v := L'.v + L''.v; L'.s = 0; L''.s = -L''.l$
2	$L' \rightarrow L'' B$	$L'.v := L''.v + B.v; L'.l := L''.l + 1; B.s := L'.s; L''.s := L'.s + 1$
3	$L \rightarrow B$	$L.v := B.v; L.l := 1; B.s := L.s$
4	$B \rightarrow 0$	$B.v := 0; B.s := 0.s;$
5	$B \rightarrow 1$	$B.v := 2^{\uparrow B.s}, B.s := 1.s;$

Атрибуты:

B: B.v, B.s

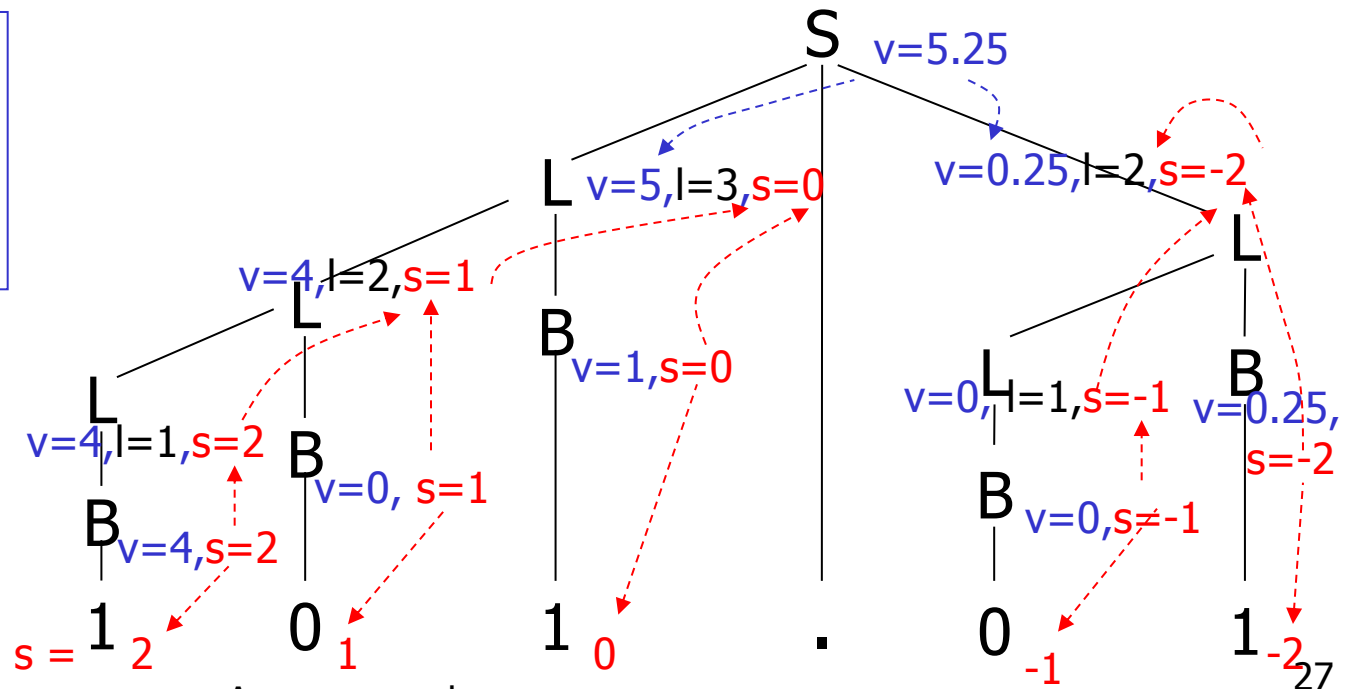
L: L.v, L.l, L.s

S: S.v

B.s – 'масштаб' B

L.l – 'длина' L

Наследуемые атрибуты – через атрибуты предков





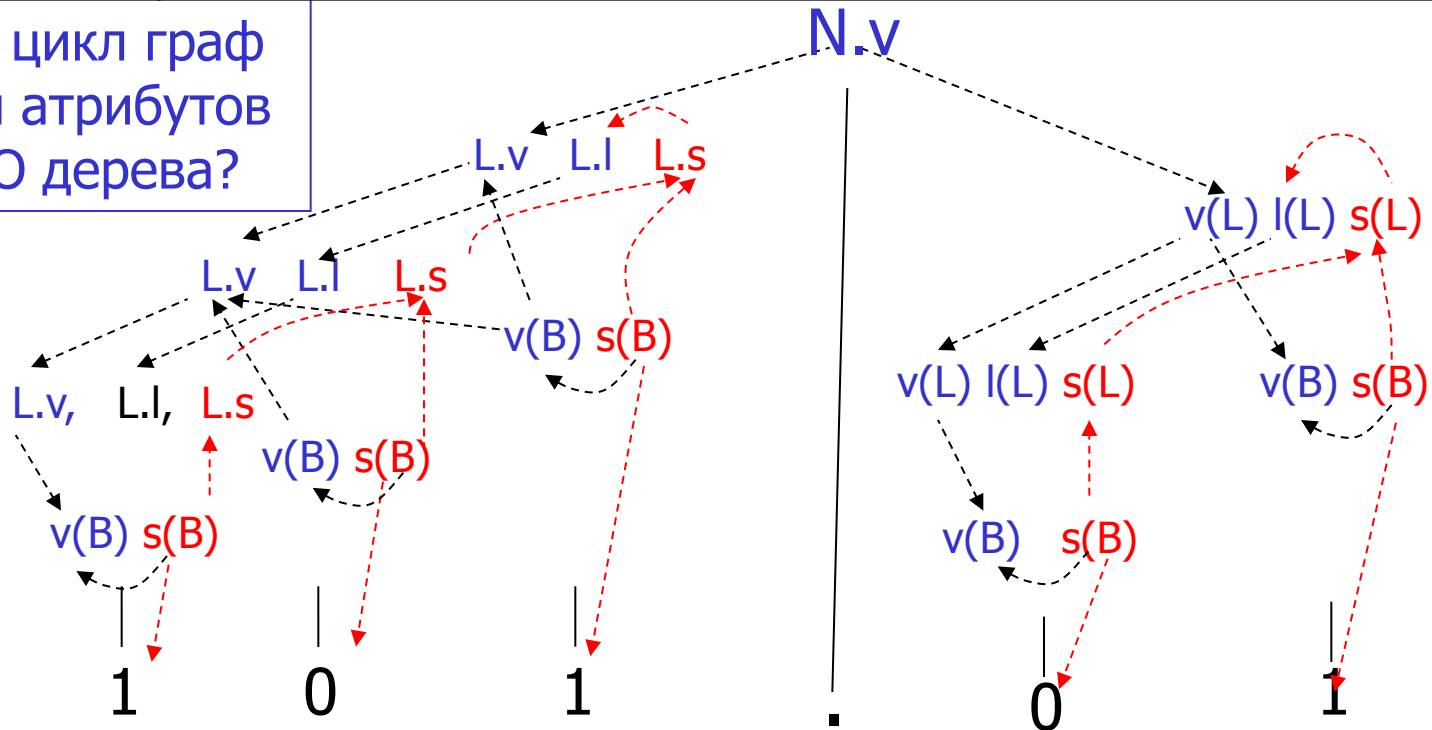
# Общие правила построения семантики

- Каждому терминалу и нетерминалу приписываются семантические атрибуты, принимающие значения из произвольного множества
  - Множество атрибутов, приписанных конкретному символу грамматики, разбиваются на два класса: синтезированные и унаследованные
  - Начальный нетерминал не может иметь унаследованных атрибутов
  - Терминальные символы не могут иметь синтезированных атрибутов
- Для каждого правила грамматики задаются локальные зависимости атрибутов символов, входящих в правило
- Для входной цепочки строится синтаксическое дерево (алгоритм синтаксического анализа). По дереву вычисляются значения всех атрибутов
  - Синтезируемые атрибуты вычисляются по дереву снизу вверх, от потомков к предкам, наследуемые атрибуты вычисляются по дереву сверху вниз, от предков к потомкам, в соответствии с определением зависимостей атрибутов для каждого правила грамматики
  - Вычисленные атрибуты корня дерева представляют собой "значение", соответствующее данному дереву вывода
- Семантические правила заданы корректно, если в любом дереве вывода эти правила позволяют вычислить все атрибуты всех узлов
- Поскольку деревьев вывода бесконечно много (!!), важно уметь определять по самой грамматике, являются ли корректными ее семантические правила

# Проверка корректности атрибутивной семантики

n	Синтаксич.правила	Семантические правила
1	$S \rightarrow L' . L''$	$N.v := L'.v + L''.v; L'.s = 0; L''.s = - L''.l$
2	$L' \rightarrow L'' B$	$L'.v := L''.v + B.v; L'.l := L''.l + 1; B.s := L'.s; L''.s := L'.s + 1$
3	$L \rightarrow B$	$L.v := B.v; L.l := 1; B.s := L.s$
4	$B \rightarrow 0$	$B.v := 0; B.s := 0.s;$
5	$B \rightarrow 1$	$B.v := 2^{\uparrow} B.s, B.s := 1.s;$

Содержит ли цикл граф зависимостей атрибутов для ЛЮБОГО дерева?



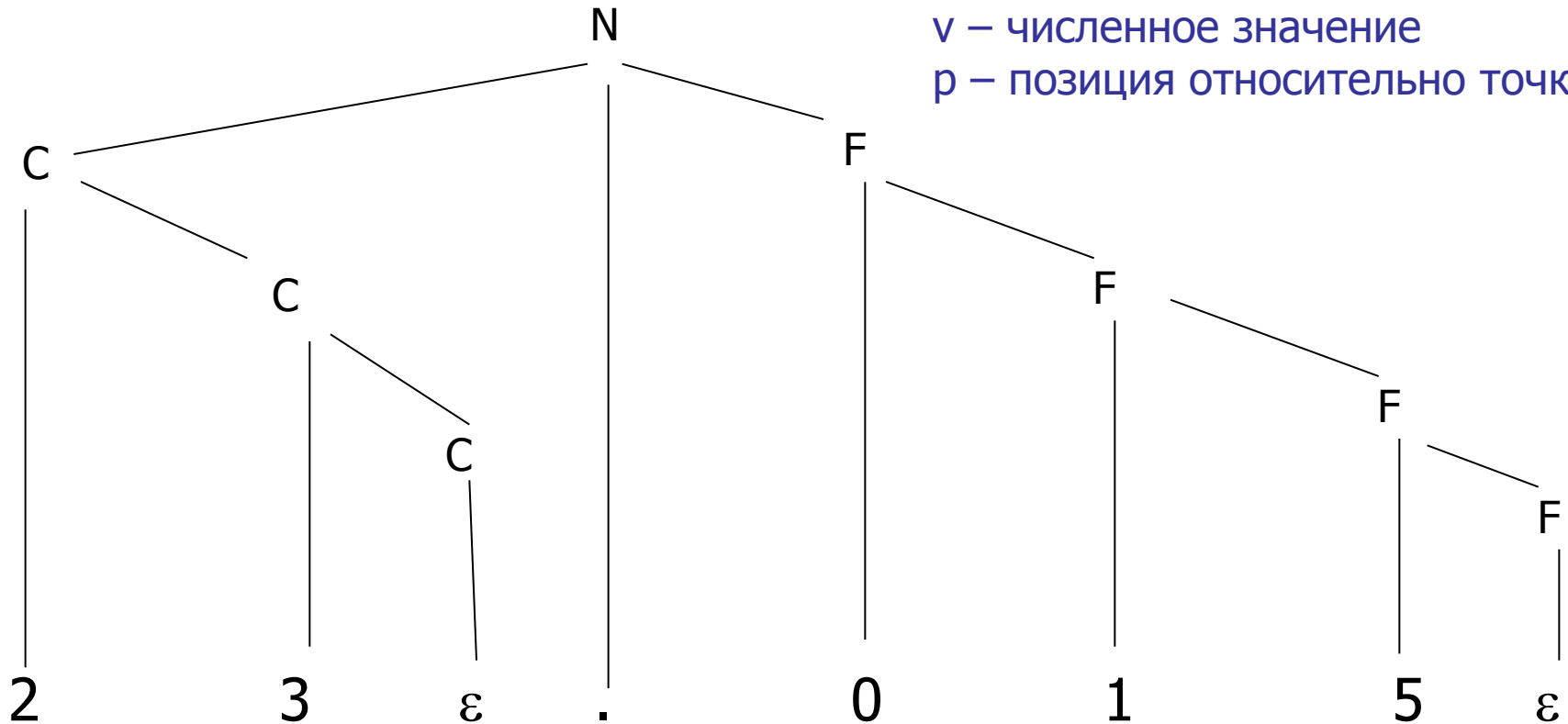
# Пример описания атрибутивной грамматики чисел

$N ::= C \cdot F$   
 $C ::= \varepsilon \mid d C$   
 $F ::= \varepsilon \mid d F$

Семантические атрибуты:

$N \quad :: \text{float } N.v$   
 $F \quad :: \text{float } F.v, \quad \text{int } F.p$   
 $C \quad :: \text{float } C.v, \quad \text{int } C.p$   
 $d \quad :: \text{int } d.v$

$v$  – численное значение  
 $p$  – позиция относительно точки



# Пример описания атрибутивной грамматики чисел

## Грамматика:

$N \rightarrow C \cdot F$   
 $C \rightarrow \varepsilon$   
 $C \rightarrow d C$   
 $F \rightarrow \varepsilon$   
 $F \rightarrow d F$

## Семантические атрибуты:

$N \quad :: \text{float } N.v \quad \blacksquare$   
 $F \quad :: \text{float } F.v, \quad \blacksquare \text{ int } F.p \quad \blacksquare$   
 $C \quad :: \text{float } C.v, \quad \blacksquare \text{ int } C.p \quad \blacksquare$   
 $d \quad :: \text{int } d.v \quad \blacksquare$

## Р - Номер позиции

$\begin{matrix} \text{Ц} & \text{Ц} & \text{Ц} & \text{Ц} & \blacksquare & \text{Ц} & \text{Ц} & \text{Ц} & \text{Ц} \\ 3 & 2 & 1 & 0 & & -1 & -2 & -3 & -4 \end{matrix}$

n	Синтаксические правила	Семантические правила
1	$N ::= C \cdot F$	$N.v := C.v + F.v;$ $F.p := -1; \text{ (наследуемый)}$
2	$C ::= \varepsilon$	$C.v := 0;$ $C.p := 0;$
3	$C ::= d C'$	$C.v := d.v * 10^{\uparrow C'.p} + C'.v;$ $C.p := C'.p + 1;$
4	$F ::= \varepsilon$	$F.v := 0$
5	$F ::= d F'$	$F.v := d.v * 10^{\uparrow F'.p} + F'.v$ $F'.p := F.p - 1 \text{ (наследуемый)}$

$$N \rightarrow C'F$$
$$\mathbb{C} \rightarrow \mathfrak{E}$$
$$C \rightarrow d C$$
$$F \rightarrow \varepsilon$$
$$F \rightarrow d F$$

N :: float N.v 

F :: float F.v, ■ int F.p ■

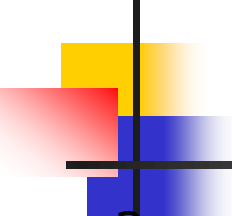
```
C :: float C.v, ■ int C.p ■
```

$$d :: \text{int } d.v \quad \blacksquare$$



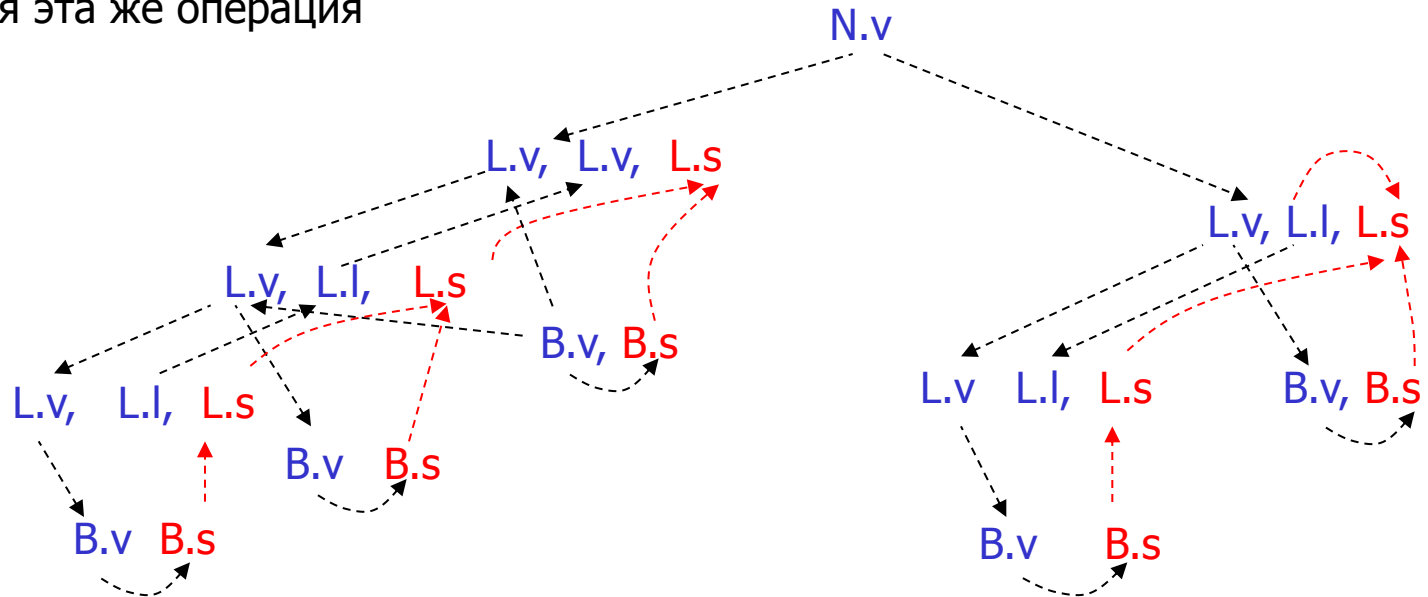

# Атрибутированное дерево разбора

- Пусть дана атрибутивная грамматика  $G$  и цепочка, принадлежащая языку, порождаемому  $G$
- Построим дерево разбора  $T$  этой цепочки в грамматике  $G$ . В дереве  $T$  каждая вершина помечена символом грамматики  $G$ . Припишем каждой вершине множество атрибутов, сопоставленных символу, которым помечена эта вершина. Дерево с сопоставленными каждой вершине атрибутами будем называть атрибутированным деревом разбора
- Между вхождениями атрибутов в дерево разбора  $T$  существуют зависимости, определяемые семантическими правилами, которые соответствуют примененным синтаксическим правилам
- Для каждого синтаксического правила  $p$  в дереве  $T$  определим  $D(p)$  - граф зависимостей атрибутов символов, входящих в правило  $p$ , как ориентированный граф, вершинами которого служат атрибуты символов, входящих в правило  $p$ , и в котором из вершины  $b(i)$  в вершину  $a(j)$  идет дуга тогда и только тогда, когда синтаксическому правилу  $p$  сопоставлено семантическое правило
$$a(j) = f_p a(j)(\dots, b(i), \dots)$$
- Граф зависимостей  $D(T)$  дерева разбора  $T$  цепочки, принадлежащей языку грамматики  $G$ , это ориентированный граф, полученный объединением графов зависимостей всех примененных в  $T$  синтаксических правил

- 
- Замещающая семантическая структура для цепочки языка строится на основе ее дерева вывода следующим образом.
  - Для каждого атрибута каждого символа в узле дерева вывода вводится своя вершина.
  - Строятся также вершины для действий, которые являются "*побочным эффектом*" семантических вычислений (как, например, действие добавление в таблицу имен  $Add(i.type, type(L))$  в семантических правилах грамматики описаний)
  - Эти "побочные вычисления" можно рассматривать как изменения глобальных семантических параметров, доступных в каждом узле дерева вывода. В вершины каждого семантического правила  $y := f(x_1, \dots, x_n)$  и каждого действия  $Act(x_1, \dots, x_m)$  в замещающей семантической структуре из вершин  $x_i$  проводится направленная дуга, показывающая отношение зависимости. Этот граф зависимости определяет порядок вычислений семантических атрибутов и выполнения семантических действий при трансляции цепочки языка

# Алгоритм топологической сортировки

- Направленный граф зависимостей вычисления семантических атрибутов в синтаксическом дереве (если он не содержит циклов) представляет собой граф отношения частичного порядка (диаграмму Хассе)
- Для вычисления значений семантических атрибутов используется алгоритм топологической сортировки: берется любая висячая вершина, которая ни от чего не зависит, и выкидывается из графа. Она уже имеет значение. В оставшемся графе выполняется эта же операция

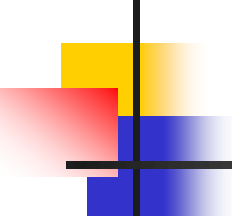


Атрибутные грамматики позволяют структурировать семантику, т.е. разделить сложность выполнения семантических операций на множество простых функций, упорядоченных по синтаксическим правилам



# Классы атрибутных грамматик

- Для каждого класса свой метод построения транслятора и свои ограничения на зависимости между атрибутами
  - Самый общий класс имеет только одно ограничение – чтобы в зависимостях между атрибутами любого построенного синтаксического дерева не было циклов. Тогда можно построить синтаксическое дерево, найти все зависимости между его атрибутами, отсортировать эти зависимости с помощью топологической сортировки и выполнить вычисления в найденном порядке. Обычно много вычислений
  - **S-атрибутные грамматики.** Если в грамматике существуют только синтезируемые атрибуты, то их можно легко вычислить в процессе либо нисходящего, либо восходящего синтаксического анализа
  - **L-атрибутные грамматики.** Если атрибуты символа  $X$  зависят от атрибутов символов, находящихся только слева от него в синтаксическом правиле, то циклов в зависимостях быть не может, а все атрибуты можно вычислить в процессе синтаксического **нисходящего** анализа. L-атрибутные грамматики использует компилятор компиляторов Coco-R
  - **LR-атрибутные грамматики.** Если атрибуты последовательно можно вычислить в процессе **восходящего** синтаксического анализа, то это LR-атрибутная грамматика. Это подкласс L-атрибутных грамматик



---

## Двусмысленные (неоднозначные) грамматики

# Грамматика операторов цикла языка Милан

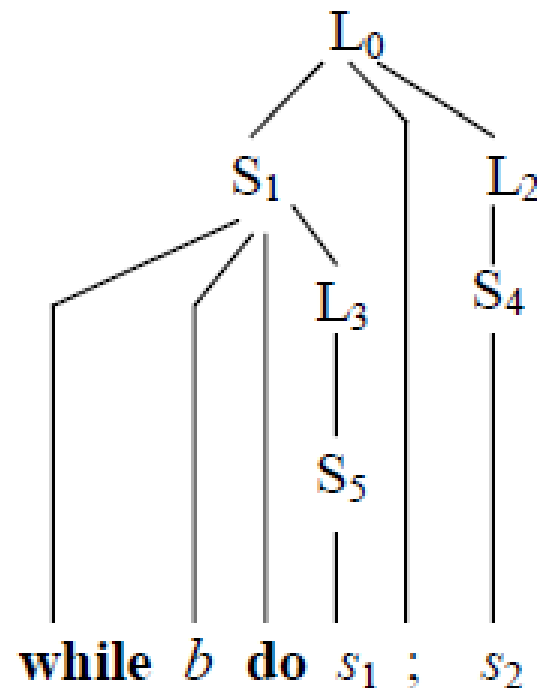
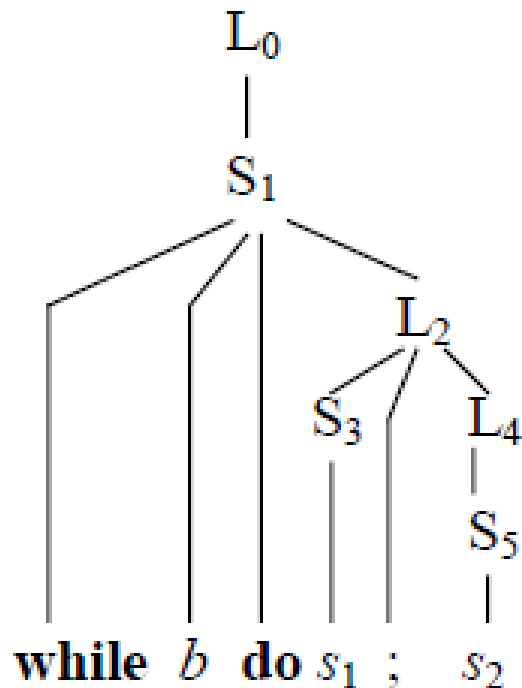
Операторы цикла языка Милан порождаются следующей грамматикой:

$G_{\text{while}}: L \rightarrow S \mid S;L$

$S \rightarrow \textbf{while } b \textbf{ do } L \mid s$

Здесь мы абстрагировались от структуры условия (обозначив любое условие терминалом  $b$ ) и от структур всех типов операторов, кроме оператора цикла (обозначив их все терминалом  $s$ )

Как понимать цепочку `while b do  $s_1$ ;  $s_2$`  (??)



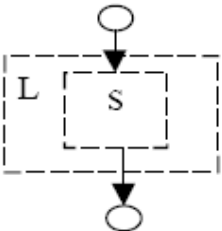
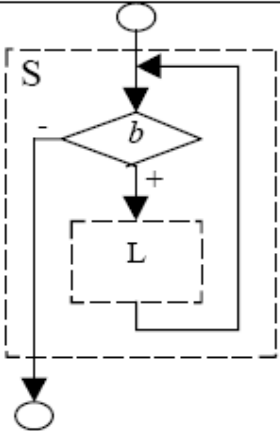
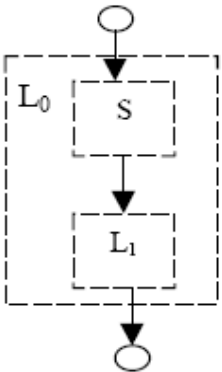
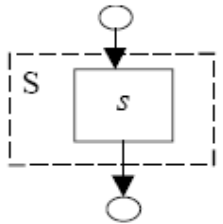


## Причины возникновения неоднозначности

---

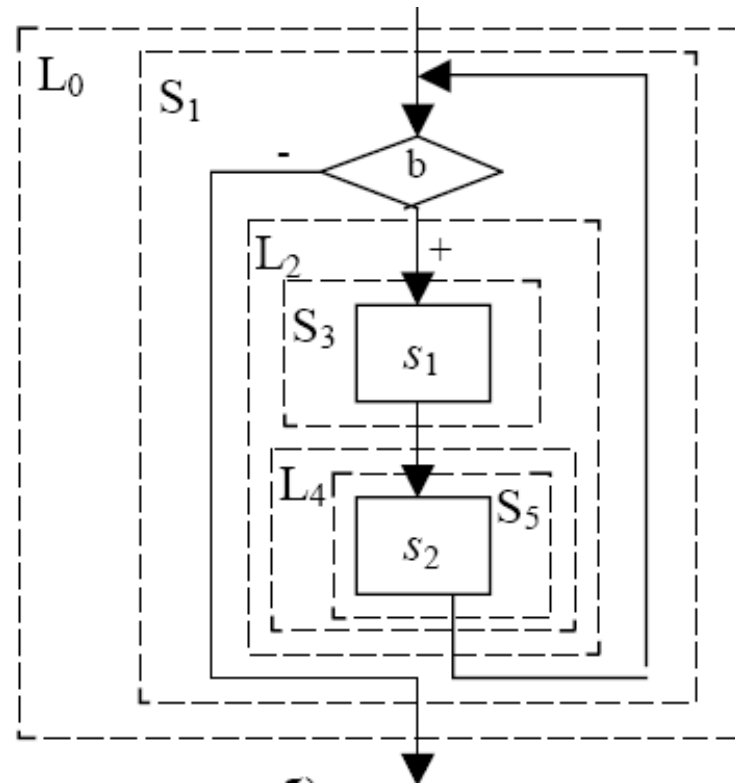
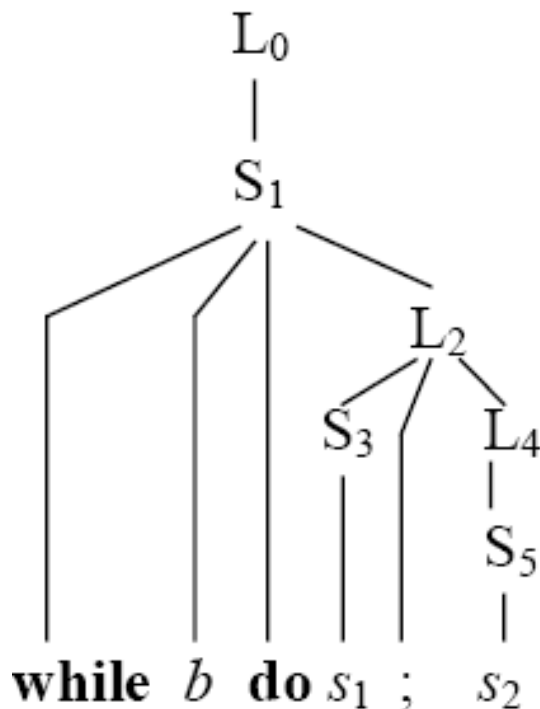
- Поскольку семантические вычисления основываются именно на дереве вывода, очевидно, что различные деревья вывода могут привести к различным интерпретациям этой цепочки
- Пусть семантический параметр, сопоставленный каждому нетерминалу грамматики – это блок последовательных вычислений с одним входом и одним выходом. Терминалу  $s$  сопоставим простейший блок (например, будем его интерпретировать, как оператор присваивания)
- Тогда каждому синтаксическому правилу (продукции) можно очевидным образом сопоставить семантические правила получения структуры сложного блока (соответствующего левой части продукции) из более простых

# Атрибутная грамматика операторов цикла

N	Продукции	Семантика продукций	Пояснения	N	Продукци и	Семантика продукций	Пояснения
1.	$L \rightarrow S$		Последовательность операторов L – это блок S.	3.	$S \rightarrow \text{while } b \text{ do } L$		Блок S определяет циклически повторяющееся выполнение блока L до тех пор, пока b не станет false.
2.	$L_0 \rightarrow S ; L_1$		Блок $L_0$ – это последовательное соединение блока S и блока $L_1$ .	4.	$S \rightarrow s$		Блок S – это простейший блок s.



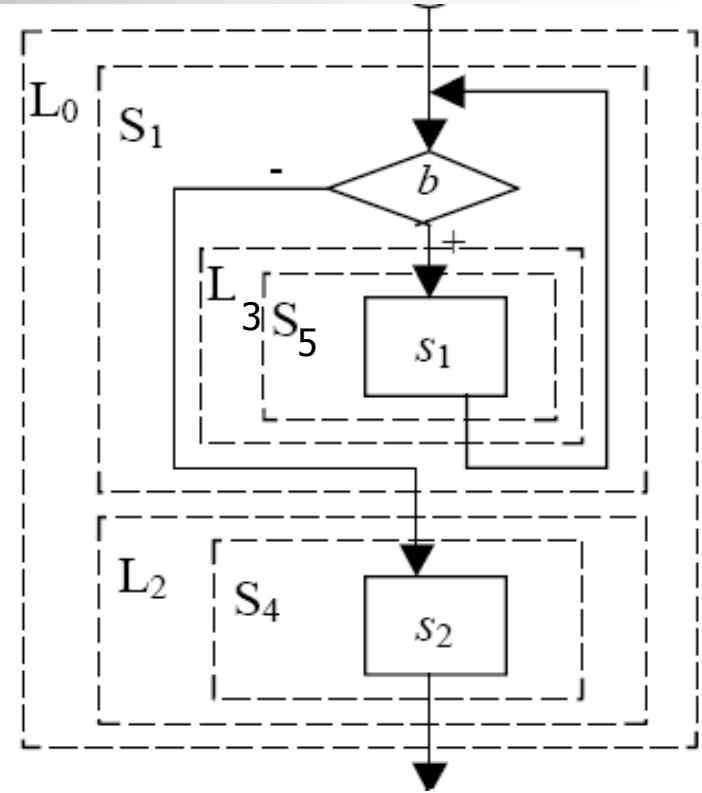
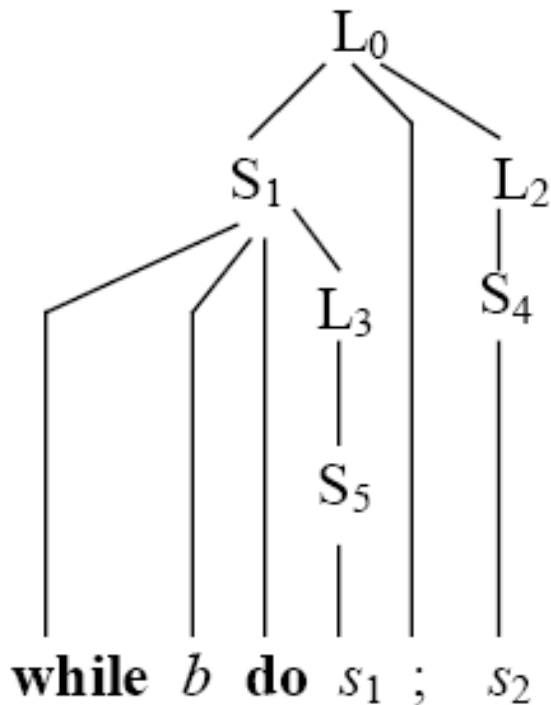
## Две интерпретации цепочки '**while** *b do* *s1*; *s2*'



Вся конструкция  $L_0$  - это один единственный оператор цикла  $S_1$ , а входящая в этот оператор цикла последовательность операторов  $L_2$  состоит из оператора  $S_3$  и еще одной последовательности операторов  $L_4$

Оператор  $S_3$  - это простейший оператор  $s_1$ , указанный в цепочке. В свою очередь, последовательность операторов  $L_4$  состоит из одного оператора  $S_5$ , который является простейшим оператором  $s_2$  анализируемой цепочки

# Две интерпретации цепочки 'while b do s1;s2 '



Вторая интерпретация этой же цепочки определяет всю конструкцию  $L_0$  как пару операторов: один из них – это оператор цикла  $S_1$  и следующая за ним последовательность операторов  $L_2$ . В операторе цикла  $S_1$  циклически повторяется выполнение блока  $L_3$  до тех пор, пока  $b$  не станет false. Этот блок  $L_3$  состоит из одного оператора  $S_5$ , который является простейшим оператором  $s_1$  исходной цепочки. Блок  $L_2$  состоит из одного оператора  $S_4$ , который является простейшим оператором  $s_2$  анализируемой цепочки

## Устранение неоднозначности

Для того, чтобы в грамматике  $G_{\text{while}}$  избавиться от такой неоднозначности, можно изменить синтаксис: ввести правило обязательного "закрытия скобок" после оператора цикла. А именно, понимая служебное слово **do** как открывающую скобку, введем **od** как закрывающую:

$G'_{\text{while}}$ :  $L \rightarrow S \mid S ; L$   
 $S \rightarrow \text{while } b \text{ **do** } L \text{ **od** } \mid s$

Будет ли однозначной эта новая грамматика?      ДА!

Однако в общем случае проблема определения того, является ли КС-грамматика однозначна (т.е. что любая цепочка языка, порождаемого этой грамматикой, имеет только одно единственное дерево вывода), является неразрешимой



## Грамматика условных операторов

Похожая ситуация с грамматикой, задающий условный оператор в языке Милан. Для простоты будем считать, что в условном операторе может встретиться только оператор, а не список операторов. Соответствующая грамматика определяется так:

$G_{if}: \quad S \rightarrow \text{if } b \text{ then } S \text{ else } S \mid \text{if } b \text{ then } S \mid s$

Как мы показали, эта грамматика двусмысленна. Например, для цепочки:

if  $b_1$  then if  $b_2$  then  $s_1$  else  $s_2$

можно построить два дерева вывода. Эти деревья вывода определяют различную интерпретацию этой цепочки в соответствии с семантическими правилами. В соответствии с ним вся цепочка, рассматриваемая как один оператор  $S_0$ , представляет собой полный условный оператор

if  $b_1$  then  $S_1$  else  $S_2$

Оператор  $S_1$  в этом операторе, в свою очередь, является сокращенным условным оператором if  $b_2$  then  $S_3$ , в то время как операторы  $S_2$  и  $S_3$  – это простейшие атомарные в данной грамматике операторы  $s_1$  и  $s_2$  соответственно.

# Как понимать условный оператор?

*if x>0 then if y>0 then if z=0 then x++ else if y=0 then y++ else if x=0 then z++ else x:=y+2.* Запишем эту цепочку тремя способами:

*x=y=z=0*

```
if x>0 then
  if y>0 then
    if z=0 then
      x++
    else
      if y=0 then
        y++
      else
        if x=0 then
          z++
        else
          x:=y+2
```

*x=2, y=0, z=0*

*x=y=z=0*

```
if x>0 then
  if y>0 then
    if z=0 then
      x++
    else
      if y=0 then
        y++
      else
        if x=0 then
          z++
        else
          x:=y+2
```

*x=0, y=0, z=0*

*x=y=z=0*

```
if x>0 then
  if y>0 then
    if z=0 then
      x++
    else
      if y=0 then
        y++
      else
        if x=0 then
          z++
        else
          x:=y+2
```

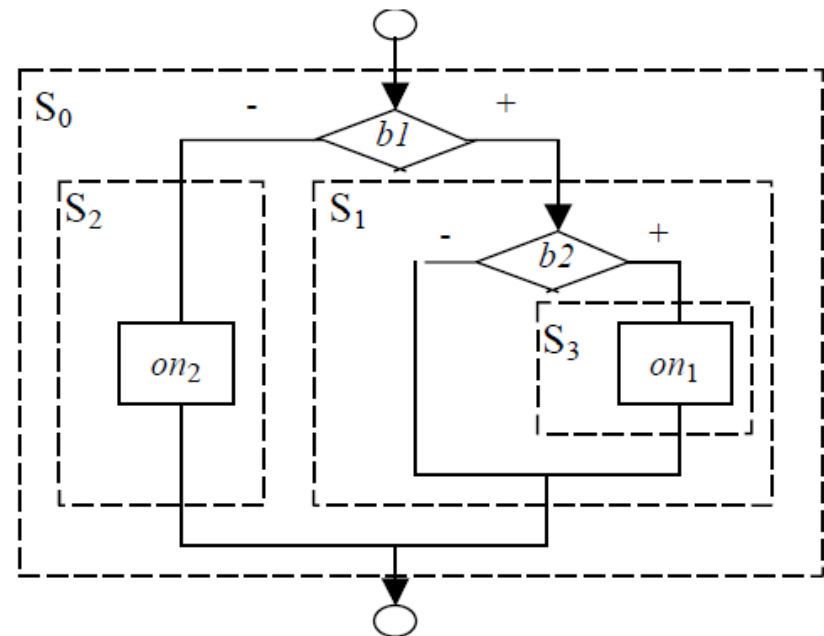
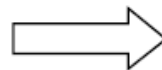
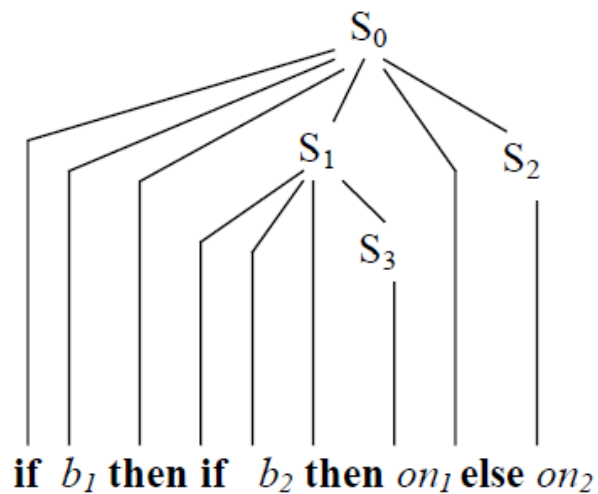
*x=0, y=1, z=0*

Цепочка, порожденная грамматикой  $G_{if}$  является двусмысленной.

**Двусмысленные конструкции – источник ошибок**

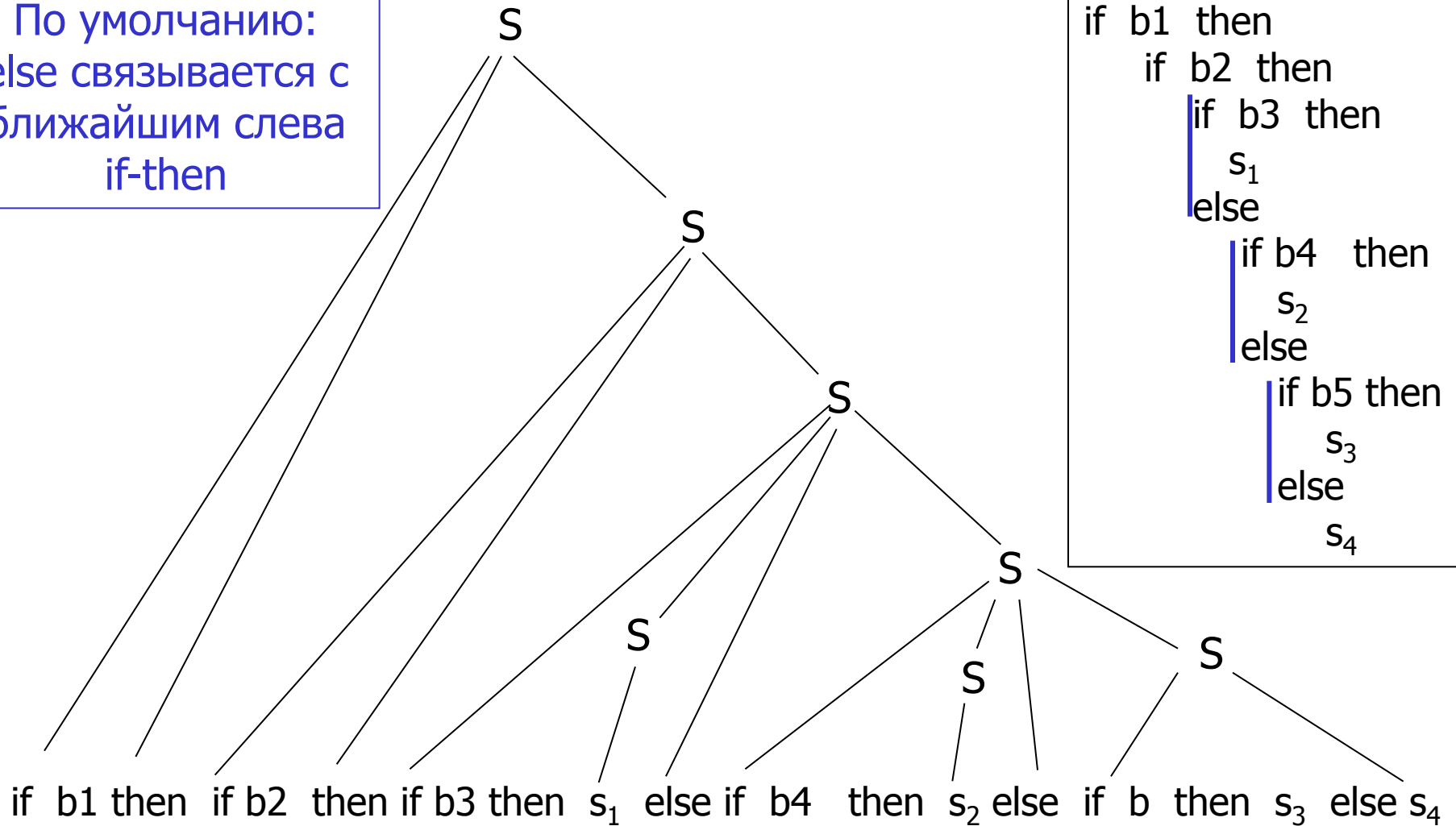
N	Продукции	Семантика продукций	Пояснения
1.	$S_0 \rightarrow \text{if } b \text{ then } S_1 \text{ else } S_2$	<pre> graph TD     Start(( )) --&gt; B{b}     B -- "+" --&gt; S1[S1]     B -- "-" --&gt; S2[S2]     S1 --&gt; Join(( ))     S2 --&gt; Join     Join --&gt; End(( ))     subgraph S0 [S0]         B         S1         S2         Join     end </pre>	<p>Выполнение оператора <math>S_0</math> строится как выполнение оператора <math>S_1</math>, если условие <math>b</math> истинно, либо оператора <math>S_2</math> в противном случае.</p>
2.	$S_0 \rightarrow \text{if } b \text{ then } S_1$	<pre> graph TD     Start(( )) --&gt; B{b}     B -- "+" --&gt; S1[S1]     B -- "-" --&gt; Join(( ))     S1 --&gt; Join     Join --&gt; End(( ))     subgraph S0 [S0]         B         S1         Join     end </pre>	<p>Выполнение оператора <math>S_0</math> строится как выполнение оператора <math>S_1</math>, если условие <math>b</math> истинно, либо пустого оператора, если условие <math>b</math> ложно.</p>
3.	$S \rightarrow on$	<pre> graph TD     Start(( )) --&gt; On[on]     On --&gt; End(( ))     subgraph S [S]         On     end </pre>	<p>Оператор <math>S</math> – это простейший оператор <i>on</i></p>

Одно из возможных деревьев вывода цепочки  
if  $b_1$  then if  $b_2$  then  $on_1$  else  $on_2$



```
if b1 then if b2 then if b3 then s1 else if b4 then s2 else if b5 then s3 else s4
```

По умолчанию:  
else связывается с  
ближайшим слева  
if-then



```

if b1 then
  if b2 then
    if b3 then
       $s_1$ 
    else
      if b4 then
         $s_2$ 
      else
        if b5 then
           $s_3$ 
        else
           $s_4$ 

```



## Устранение неоднозначности: изменим грамматику

$S \rightarrow \text{if } b \text{ then } S \mid \text{if } b \text{ then } S \text{ else } S \mid s$

else  $\rightarrow$  к ближайшему if-then

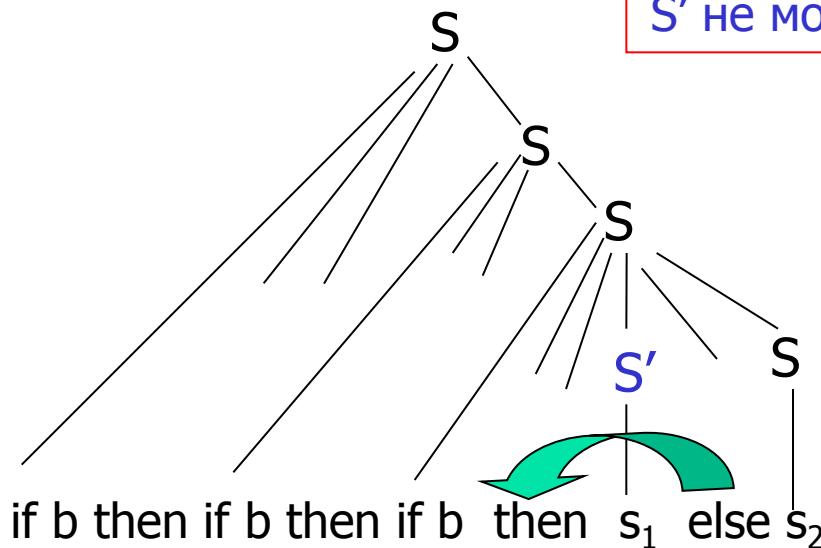
Однозначная грамматика:

$S \rightarrow \text{if } b \text{ then } S \mid \text{if } b \text{ then } S' \text{ else } S \mid s$

$S' \rightarrow \text{if } b \text{ then } S' \text{ else } S \mid s$

После then до else не  
может встретиться  
короткий условный  
оператор!

$S'$  не может быть коротким!



## Устранение неоднозначности: изменим язык

Возможны различные решения, позволяющие сделать грамматику условных операторов однозначной. Наиболее простое и очевидное – потребовать "закрывающей скобки" после условного оператора. Таким служебным словом, может быть либо `end`, либо `fi` – симметрично "открывающей скобке" `if`. Такая грамматика, в которой окончание условного оператора будет фиксировано, будет недвусмысленной даже если разрешить использовать последовательность операторов после `then` и `else`:

по аналогии с

(	)
[	]
{	}
do	od
case	esac
if	fi

$S \rightarrow \text{if } b \text{ then } S \text{ else } L \text{ **fi** } \mid \text{if } b \text{ then } S \text{ **fi** } \mid \text{on}$

# Измененная грамматика языка Милан

- Изменяем язык! Добавляем закрывающие скобки в while-do и if-then
- Стала ли грамматика Милана однозначной???

```
begin
  m:=read;
  n:=read;
  while m≠n do
    if m>n then m:=m-n
              else n:=n-m
    fi
  od;
  write(m)
end
```

$G^1$ :  $\text{Prog} \rightarrow \underline{\text{begin}} \ L \ \underline{\text{end}}$   
 $L \rightarrow S \mid S ; L$   
 $S \rightarrow i := E \mid$   
      $\underline{\text{if}} \ B \ \underline{\text{then}} \ L \ \underline{\text{fi}} \mid$   
      $\underline{\text{if}} \ B \ \underline{\text{then}} \ L \ \underline{\text{else}} \ L \ \underline{\text{fi}} \mid$   
      $\underline{\text{while}} \ B \ \underline{\text{do}} \ L \ \underline{\text{od}} \mid$   
      $\underline{\text{output}} \ (E)$   
 $B \rightarrow E \ q \ E$   
 $E \rightarrow (E) \mid E \ \underline{\text{addop}} \ E \mid E \ \underline{\text{mulop}} \ E \mid i \mid c \mid \underline{\text{read}}$

- Все терминалы – лексемы

begin – служебное слово begin, end – служебное слово end, ...

*i* – имя переменной (какая конкретно переменная – в индексе)

*q* – символ отношения (какое конкретно отношение – в индексе)

*c* – константа (какая конкретно константа – в индексе)

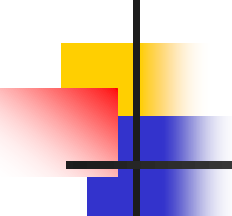
addop – операция типа сложения (индекс 0 – операция '+', 1 – операция '-')

mulop – операция типа умножения (индекс 0 – операция '\*', 1 – операция '/')<sub>52</sub>



# Проблемы неоднозначных грамматик

- Примеры неоднозначных грамматик построить легко. Например, введем в КС-грамматику правило  $A \rightarrow A$  для какого-нибудь нетерминала  $A$
- Для некоторых языков, порождаемых неоднозначными грамматиками, можно построить однозначную грамматику. Но есть КС-языки, для которых невозможно построить однозначную грамматику. Такие языки называются существенно неоднозначными.
- **Т е о р е м а.**  
Не существует алгоритма, который для любой КС-грамматики может определить, является эта грамматика однозначной, или нет.  
Не существует алгоритма, который для любого языка может определить, существует ли для этого языка однозначная КС-грамматика, или нет
- Иными словами, проблема однозначности КС-грамматик и КС-языков алгоритмически неразрешима
- На практике из класса всех КС-грамматик выделяют подклассы таких КС-грамматик, которые недвусмысленны и позволяют выполнить синтаксический анализ простыми алгоритмами. Обычные языки высокого уровня описываются грамматиками из этих более простых классов



---

Существуют и другие подходы (менее систематические) к выполнению семантических вычислений при синтаксическом анализе, чем атрибутивные семантики

# Определение семантики ЯП через преобразование предикатов

Пусть язык включает операторы:

$x := E$

$S1; S2$

**if B then S fi**

**while B do S od**

## ПРАВИЛА СТРУКТУРНОЙ ИНДУКЦИИ в семантике Хоара:

$P \rightarrow \text{begin } L \text{ end}$

$\{R1\} L \{R2\} \Rightarrow \{R1\} P \{R2\}$

$L \rightarrow S$

$\{R1\} S \{R2\} \Rightarrow \{R1\} L \{R2\}$

$L_1 \rightarrow L_2 ; S$

$\{R1\} L_2 \{R3\} \& \{R3\} S \{R2\} \Rightarrow \{R1\} L_1 \{R2\}$

$S \rightarrow x := E$

$\{R1 \Rightarrow R2 \mid x \leftarrow E\} \Rightarrow \{R1\} S \{R2\}$

$S \rightarrow \text{if } B \text{ then } L \text{ fi}$

$[\{R1 \wedge B\} L \{R2\}] \& [R1 \wedge \neg B \Rightarrow R2] \Rightarrow \{R1\} S \{R2\}$

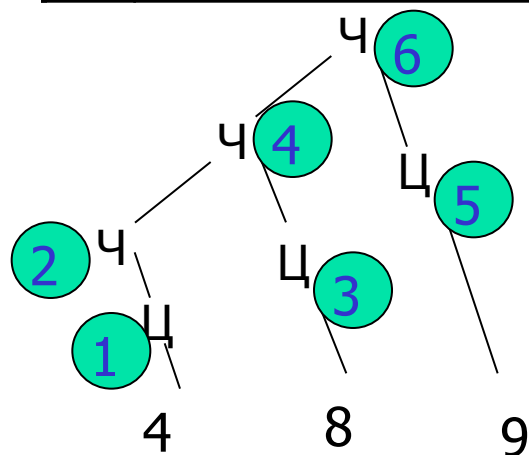
$S \rightarrow \text{while } B \text{ do } L \text{ od}$

$[\{R1 \wedge B\} L \{R1\}] \& [R1 \wedge \neg B \Rightarrow R2] \Rightarrow \{R1\} S \{R2\}$

# Транслирующие грамматики

Транслирующие грамматики переводят любую цепочку языка в последовательность семантических действий, выполнение которых представляет трансляцию этой цепочки в требуемый результат

N	Правило грамматики	Семантическое действие
1	$\langle \text{Число} \rangle \rightarrow \langle \text{Цифра} \rangle$	$y_1$ : "считай это значение результатом"
2	$\langle \text{Число} \rangle \rightarrow \langle \text{Число} \rangle \langle \text{Цифра} \rangle$	$y_2$ : "добавь его к предыдущему результату, умноженному на 10"
$3_0$	$\langle \text{Цифра} \rangle \rightarrow 0$	$y_{30}$ : "возьми значение 0"
$3_i$		...
$3_9$	$\langle \text{Цифра} \rangle \rightarrow 9$	$y_{39}$ : "возьми значение 9"



При порядке обхода или построения дерева *Левосторонне-Правосторонне-Корень* порядок выполнения семантик:

$y_{34}$ ;  $y_1$ ;  $y_{38}$ ;  $y_2$ ;  $y_{35}$ ;  $y_2$  – операционные символы

"Возьми значение 4, считай его результатом, возьми значение 8, добавь его к предыдущему результату, умноженному на 10, возьми значение 9, добавь его ..."

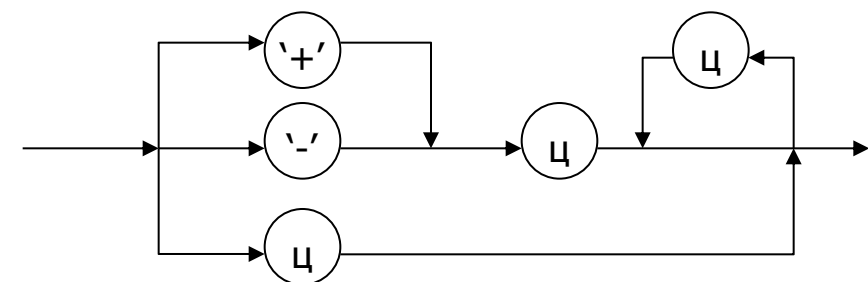
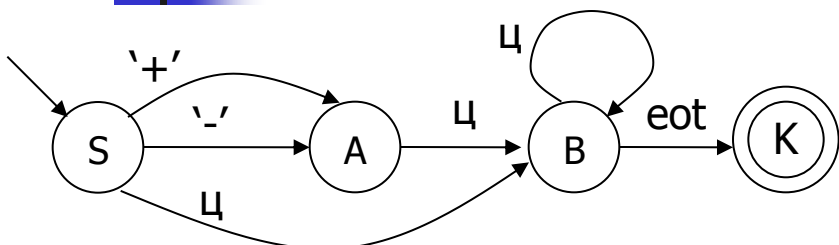
- Операционный подход к определению семантики языка высокого уровня [1] состоит в описании алгоритма интерпретации программы на этом языке в терминах некоторой абстрактной машины
- Задание операционной семантики языка программирования состоит в том, что каждое правило порождающей грамматики языка связывается с алгоритмом, порождающим фрагмент программы, соответствующий определяемой этим правилом конструкции
- При трансляции цепочки языка обход синтаксического дерева (в том или ином виде) порождает последовательность исполняемых операций

[1] Лавров С. С. Программирование. Математические основы, средства, теория. — БХВ-Петербург, 2001

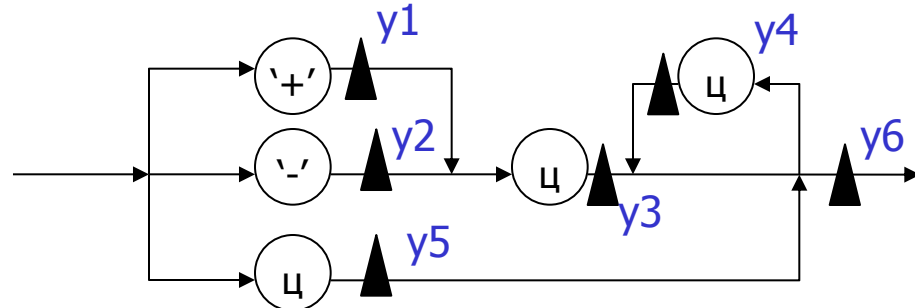
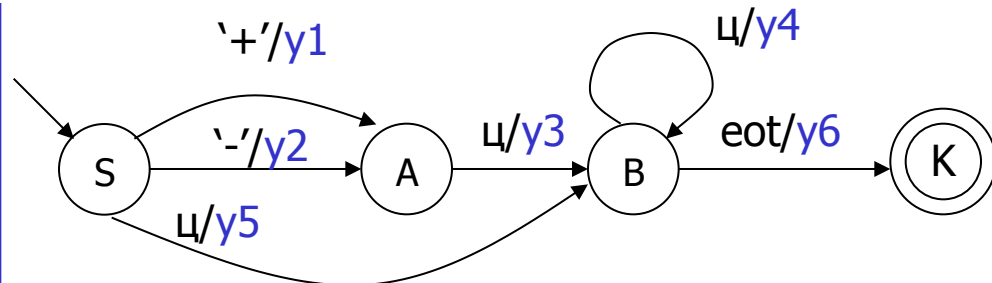


# Грамматика для задания целых констант

233; -1342;  
+7686



$S \rightarrow + A \mid -A \mid \text{ц} B$   
 $A \rightarrow \text{ц} B$   
 $B \rightarrow \text{ц} B \mid \underline{\text{eot}} K$   
 $K \rightarrow \varepsilon$



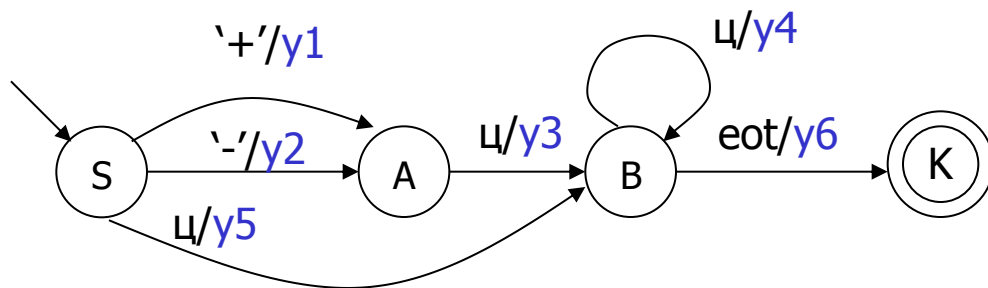
$S \rightarrow + y_1 A \mid - y_2 A \mid \text{ц} y_5 B$   
 $A \rightarrow \text{ц} y_3 B$   
 $B \rightarrow \text{ц} y_4 B \mid \underline{\text{eot}} y_6 K$   
 $K \rightarrow \varepsilon$

$y_1: x:=0; z:=1;$   
 $y_2: x:=0; z:=-1;$

$y_3: x:=\varphi(\text{ц});$   
 $y_4: x:=\varphi(\text{ц})+10*x;$

$y_5: x:=\varphi(\text{ц}); z:=1;$   
 $y_6: X:=x * z;$

## Пример трансляции цепочки



- По входу  
– 8 9 5 7 6 eot  
будет выдана цепочка семантических действий:  
 $y2, y3, y4, y4, y4, y4, y6$ .

# Операционная семантика: транслятор языка чисел римской системы счисления

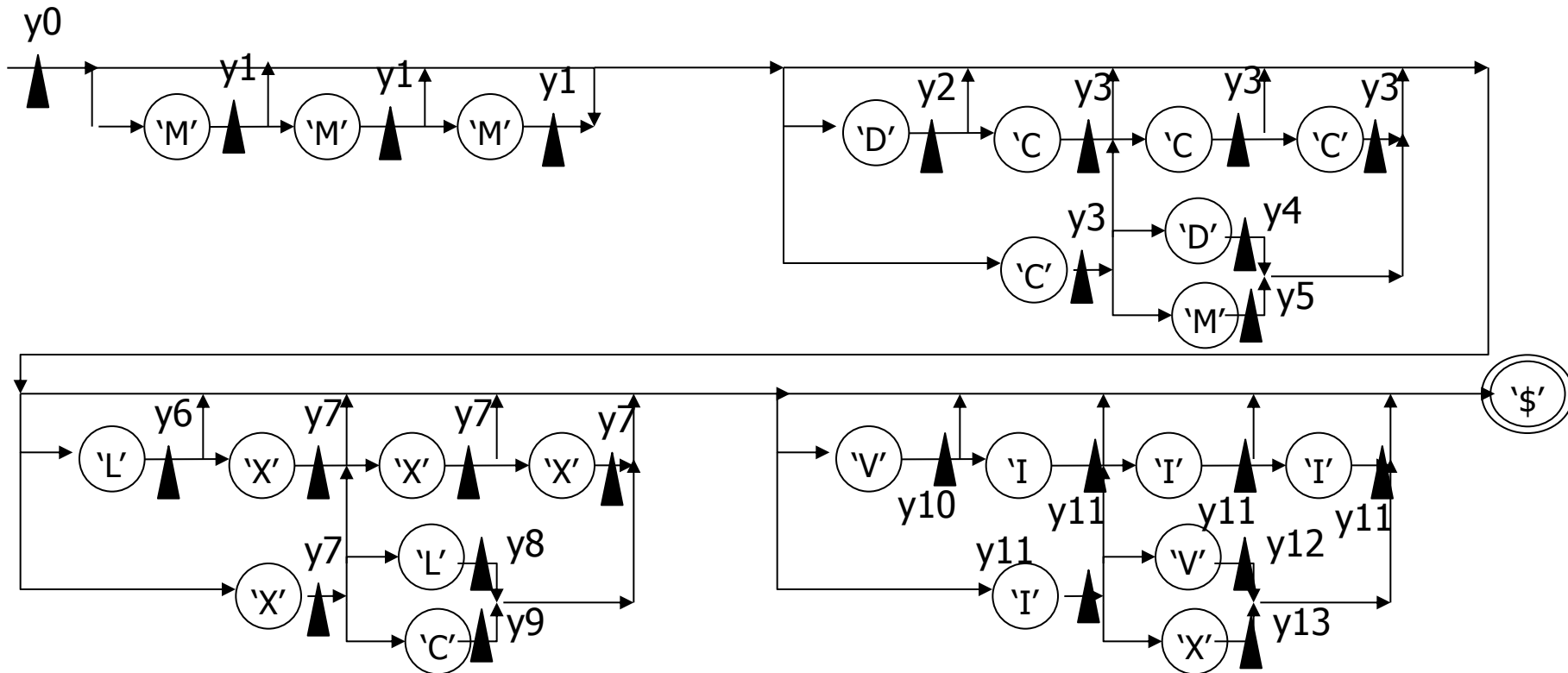
**I** = 1    **V** = 5  
**X** = 10    **L** = 50  
**C** = 100    **D** = 500  
**M** = 1000

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;



Каждая входная цепочка вызывает какую-то цепочку семантических действий:  
 C M X C V I I \$ => y0 y3 y5 y7 y9 y10 y11 y11 => 997



## Заключение

- Основная идея Хомского: понимание цепочки языка должно основываться на синтаксисе, структуре цепочки, для КС-языка – на дереве его вывода
- Метод атрибутивных трансляций Кнута состоит в том, что *"значение" цепочки КС-языка можно найти, вычислив все семантические атрибуты в дереве вывода этой цепочки*
- Атрибутная грамматика - это КС грамматика, в которой для каждого символа определен набор его атрибутов, и для каждого синтаксического правила задано множество формул, вычисляющих атрибуты
- Атрибуты могут быть самыми разными, например значение статического выражения (4 для «2+2»), таблица имен, видимых из данной конструкции, ассемблерный код, реализующий данный оператор и т.п.
- Атрибуты вычисляются по правилам, определяемым для каждой продукции грамматики. Для синтезируемых атрибутов замещающая семантическая структура будет повторять дерево вывода. Связи (зависимости одних атрибутов от других) не распространяются вне локальных зависимостей атрибутов внутри продукций грамматики
- Наследуемые атрибуты служат для передачи информации вниз по синтаксическому дереву (от корня к листьям), а синтезируемые – в обратном направлении



## Заключение (2)

---

- Использование как синтезируемых, так и наследуемых атрибутов позволяет определить семантику языка так, чтобы значение любого атрибута в любом узле могло произвольным образом зависеть от параметров и структуры всего дерева вывода
- Отсюда следует, что любая мыслимая функция, определенная на дереве вывода, может быть представлена как атрибут произвольного узла. Это доказано формально
- Атрибутные грамматики позволяют структурировать семантику, т.е. разделить сложность выполнения семантических операций на множество простых функций, упорядоченных по синтаксическим правилам
- Существуют и другие, менее систематические методы семантических вычислений при трансляции языков. Но все они, явно или неявно, используют структуру входной цепочки, получаемую на этапе синтаксического анализа

# Персоналии: Дональд Кнут

Лучший способ в чём-то разобраться до конца —научить этому компьютер



**Дональд Кнут** (р. 1938 г) известен всему мировому сообществу, имеющему дело с компьютерами

Почётный профессор Стэндфордского университета, признанный идеолог программирования, автор 19 монографий, разработчик нескольких известных программных технологий. Автор всемирно известной серии книг “**Искусство программирования**”, создатель издательских систем T<sub>E</sub>X и METAFONT для набора и вёрстки физико-математических текстов

Лауреат премии Тьюринга

- В области формальных языков и трансляции Дональд Кнут имеет фундаментальные результаты в синтаксисе и семантике. Он разработал:
  - блестящую теорию атрибутивной семантики
  - наиболее мощный метод синтаксического анализа контекстно-свободных языков – LR(k)-грамматики
- И то, и другое является сейчас основными методами, используемыми в построении практических компиляторов.
- Этим вопросам будут посвящены его новые (еще не вышедшие) книги серии “**Искусство программирования**”



---

Спасибо за внимание