Методы разработки компиляторов. Атрибутные грамматики

Лекции по курсу

Методы разработки компиляторов

семестр 7, ФИИТ

Михалкович С.С.

Атрибутные грамматики

Определения

Определение. Атрибутная грамматика – расширение КС-грамматики, при котором каждому символу (терминалу, нетерминалу) ставится в соответствие конечное множество атрибутов (семантических параметров) некоторого типа.

Для каждого правила грамматики определяется одно или несколько семантических правил – функциональных зависимостей одних семантических атрибутов от других.

Дерево разбора, включающее значения атрибутов в каждом узле, называется аннотированным или украшенным, а процесс вычисления атрибутов по дереву – аннотированием дерева разбора.

Пример

Грамматика:	Семантические действия:
$L \to E$	$print(E.val)$
$E \to E + T$	$E.val = E_1.val + T.val$
$E \to T$	$E.val = T.val$
$T \to T * F$	$T.val = T_1.val * F.val$
$T \to F$	$T.val = F.val$
$F \to (E)$	$F.val = E.val$
$F \to num$	$F.val = num.lexval$

Здесь val (значение) – атрибут вещественного типа, связываемый с нетерминалами, $lexval$ - атрибут вещественного типа, связываемый с терминалом num

Во всех правилах атрибут нетерминала левой части вычисляется как функция атрибутов символов правой части. Поэтому первое действие следует записать так:

$$L.fic = print(E.val)$$

где fic – фиктивный атрибут

Интерпретатор по атрибутной грамматике

- Используя данную атрибутную грамматику, можно производить вычисления прямо в процессе вывода. Например, для строки

$$2 * (3 + 1)$$

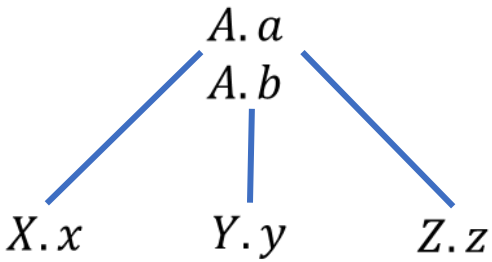
левосторонний вывод имеет вид:

$$L \Rightarrow E \Rightarrow T \Rightarrow T * F \Rightarrow F * F \Rightarrow 2 * F \Rightarrow 2 * (E) \Rightarrow 2 * (E + T) \ldots \Rightarrow 2 * (3 + 1)$$

Семантические действия делаются по дереву снизу вверх – их выполнение является *интерпретацией выражения*.

Синтезированные атрибуты

Определение. Атрибут называется *синтезированным*, если его значение в узле дерева разбора определяется значением атрибутов в дочернем узле и других атрибутов в самом узле

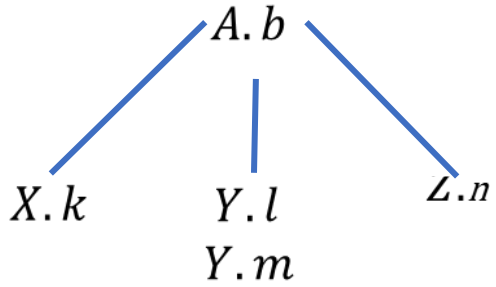


$$a = f(x, y, z, b)$$
$$b = f(x, y, z)$$

Синтезированный атрибут служит для передачи информации по дереву снизу вверх

Унаследованные атрибуты

Определение. Атрибут называется *унаследованным*, если его значение в узле дерева разбора определяется значением атрибутов в родительском узле и дочерних атрибутов по отношению к этому родительскому (братья и сёстры узла и другие атрибуты данного узла)



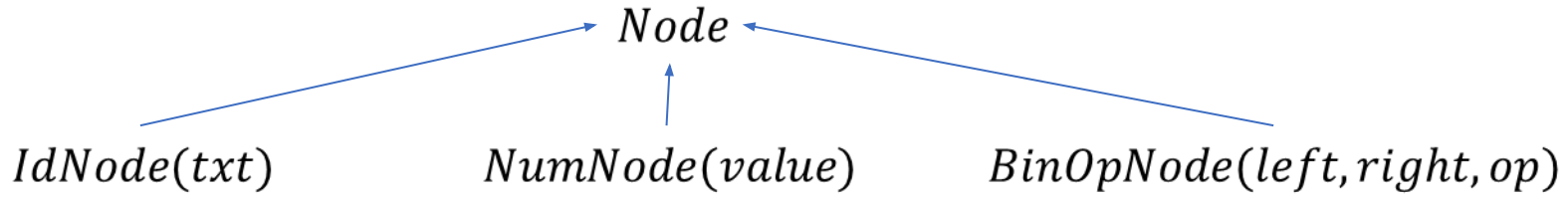
$$l = f(b, k, m, n)$$
$$m = f_1(b, k, n)$$

Унаследованный атрибут служит для передачи информации по дереву сверху вниз

Пример 2

Грамматика:	Семантические действия:
$L \to E$	$L.node = E.node$
$E \to E + T$	$E.node = new\ BinOpNode(E_1.node, T.node, '+')$
$E \to T$	$E.node = T.node$
$T \to T * F$	$T.node = new\ BinOpNode(T_1.node, F.node, '*')$
$T \to F$	$T.node = F.node$
$F \to i$	$F.node = new\ IdNode(i.txt)$
$F \to num$	$F.node = new\ NumNode(num.value)$
$F \to (E)$	$F.node = E.node$

Сопоставим L, E, T, F атрибут $node: Node$, хранящий узел AST-дерева.
Иерархия узлов:

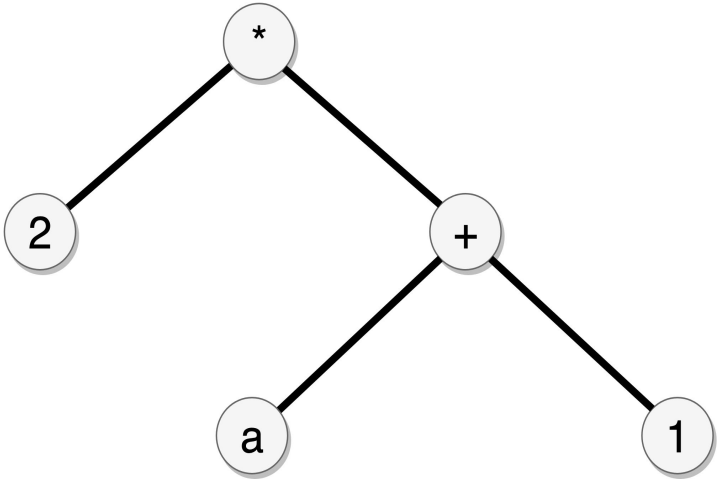


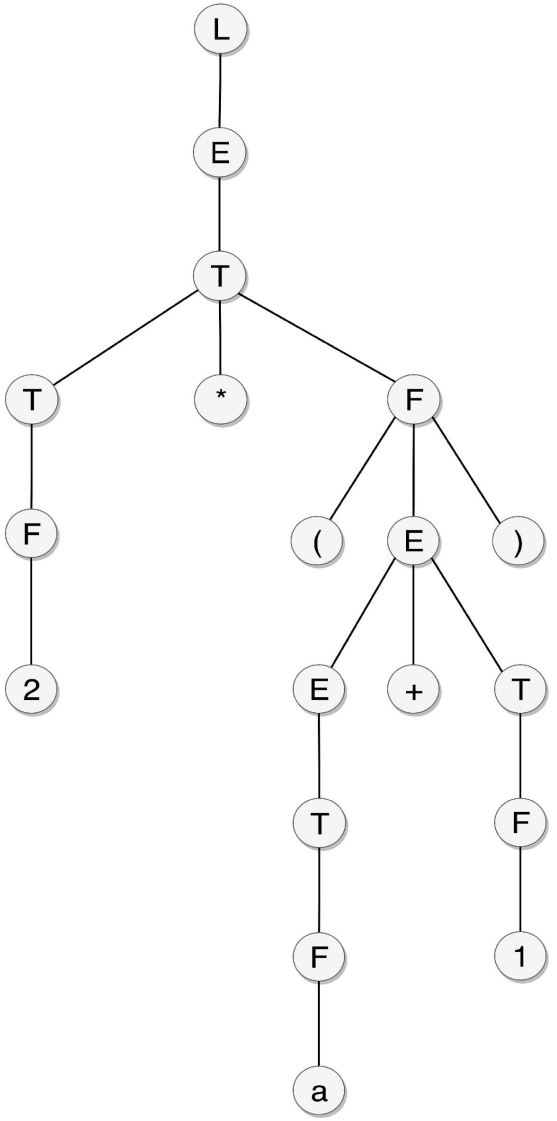
В результате вычисления всех атрибутов (синтезированных) снизу вверх мы получим AST-дерево программы.

Пример 2 – AST-дерево и дерево разбора

- Для выражения $2 * (a + 1)$

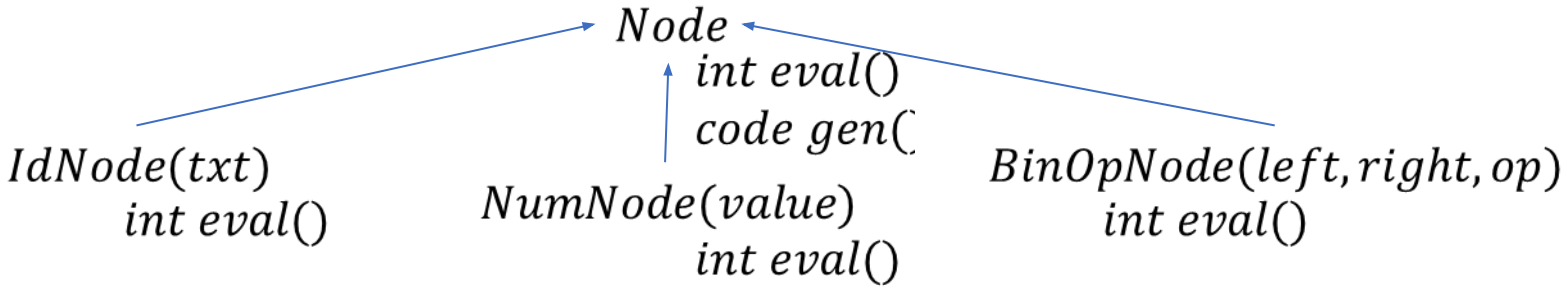
AST-дерево и дерево вывода





Пример 2 – интерпретатор

После построения AST-дерева добавим к каждому узлу метод eval() для вычисления результата поддерева



Затем вызовем eval() для корня дерева

Данный подход лучше первого интерпретатора тем, что

- Вначале строится дерево и проверяются все ошибки (компилятор в дерево)
- Можно запускать программу с различными исходными данными без перекомпиляции (это быстрее)

Пример 3. Заполнение таблицы символов по описанию переменных

- Строка

$$int\ a, b, c$$

Пример 3. Заполнение таблицы символов по описанию переменных

Грамматика:	Семантические действия:	
$D \to T\ L$	$L.type = T.type$	Унасл
$T \to int$	$T.type = integer = int.type$	Синт
$T \to double$	$T.type = real = double.type$	Синт
$L \to L_1, id$	$L_1.type = L.type$	Унасл
	$L.fic = AddSymbol(id.text, L.type)$	Синт
$L \to id$	$L.fic = AddSymbol(id.text, L.type)$	Синт

Считается, что атрибуты терминальных символов вычисляются на этапе лексического анализа и ни от чего не зависят

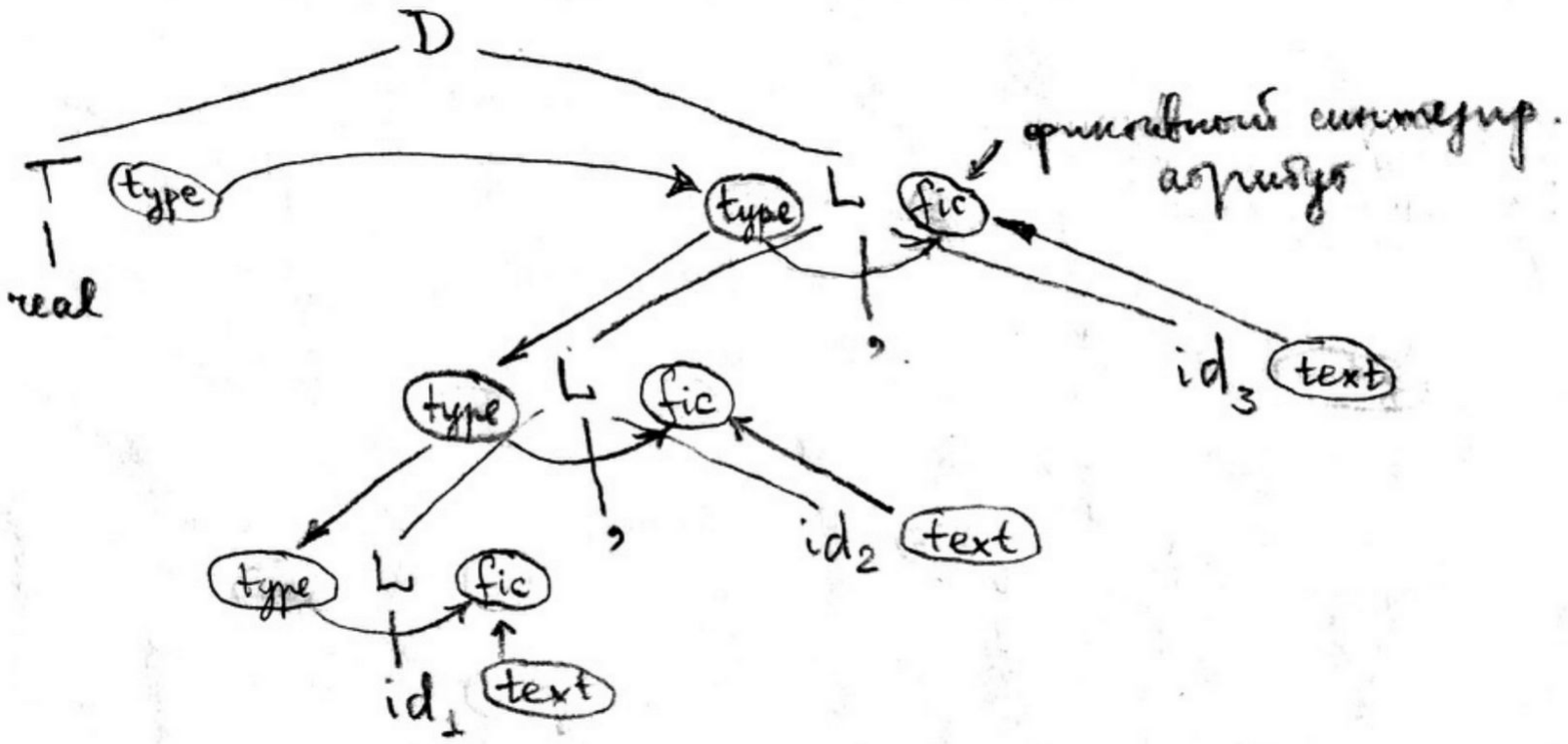
Зависимости между атрибутами в узлах дерева могут быть показаны с помощью направленного графа, называемого **графом зависимости.** Перед его использованием каждое семантическое правило приводится к виду

$$b = f(c_1, c_2, \ldots, c_n)$$

введением, если необходимо, фиктивного синтезируемого атрибута для каждого семантического правила, состоящего из вызова процедуры.

Пример 3 – дерево разбора

Для строки $real\ id_1, id_2, id_3$ дерево разбора:



Вопросы

- В каком порядке вычислять атрибуты?
- Всегда ли существует порядок, в котором можно вычислить все атрибуты?
- Можно ли предложить один алгоритм для всех деревьев разбора?

Определения, утверждения

Определение. Атрибутная грамматика называется **корректной** если для любого дерева вывода все семантические атрибуты могут быть вычислены (это определение плохое, т.к. количество деревьев вывода – бесконечное)

Утверждение. Все семантические атрибуты могут быть вычислены ⇔ граф зависимостей не имеет циклов (но граф зависимостей строится для каждого дерева вывода!!)

Алгоритм, устанавливающий корректность атрибутной грамматики непосредственно по семантическим зависимостям, построен Д. Кнутом и имеет экспоненциальную сложность относительно размеров грамматики.

Утверждение. Если все правила – синтезированные и нет циклических зависимостей между атрибутами, принадлежащими одному символу, то атрибутная грамматика корректна.

Пример 4. Грамматика для представления рациональных двоичных чисел

$$10.101 = 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-3} = 2\frac{5}{8}$$

Атрибуты

$B.v$ – значение двоичного символа (0 или 1)
$L.v$ – значение цепочки двоичных символов
$L.l$ – длина цепочки
$S.v$ – значение всего двоичного числа

Грамматика:	**Семантические действия**:	
$S \to L_1.L_2$	$S.v = L_1.v + L_2.v/2^{L_2.l}$	Синт
$L \to L_1 B$	$L.v = 2 * L_1.v + B.v$	Синт
	$L.l = L_1.l + 1$	Синт
$L \to B$	$L.v = B.v$	Синт
	$L.l = 1$	Синт
$B \to 0$	$B.v = 0$	Синт
$B \to 1$	$B.v = 1$	Синт

Пример 4. Грамматика 2 для представления рациональных двоичных чисел

- $10.101 = 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-3} = 2\frac{5}{8}$

Атрибуты

$B.v$ – значение двоичного символа

$B.s$ – вес двоичного символа (его положение относительно точки)

$L.v$ – значение подцепочки двоичных символов с учетом его положения в цепочке

$L.l$ – длина цепочки

$L.s$ – вес цепочки (положение относительно точки крайнего правого бита)

$S.v$ – значение всего двоичного числа

Грамматика:

$S \to L_1.L_2$

$L \to L_1 B$

$L \to B$

$B \to 0$

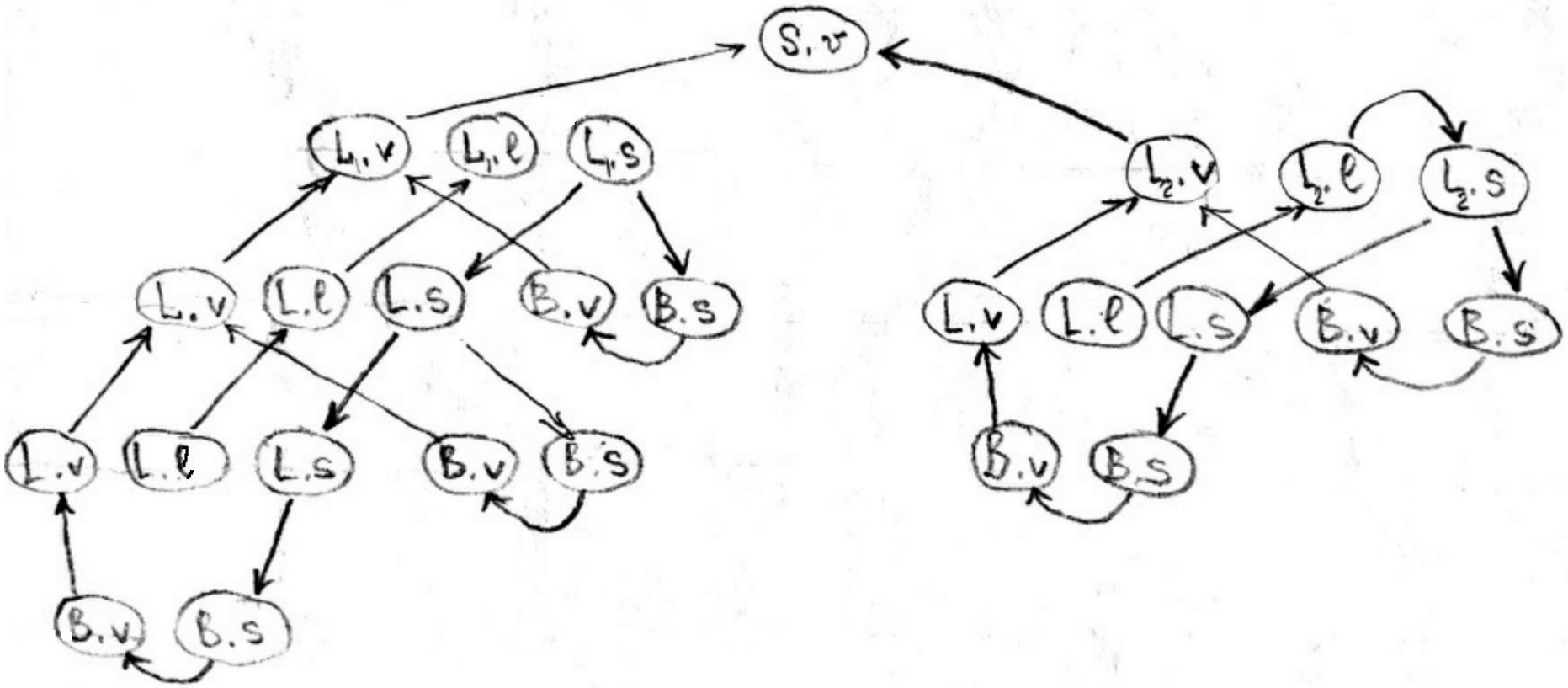
$B \to 1$

Пример 4. Грамматика 2 для представления рациональных двоичных чисел

- $10.101 = 2\frac{5}{8}$

Грамматика:	**Семантические действия**:	
$S \to L_1.L_2$	$S.v = L_1.v + L_2.v$	Синт
	$L_1.s = 0$	Синт
	$L_2.s = -L_2.l$	Синт
$L \to L_1 B$	$L.v = L_1.v + B.v$	Синт
	$B.s = L.s$	Унасл
	$L_1.s = L.s + 1$	Унасл
	$L.l = L_1.l + 1$	Синт
$L \to B$	$L.v = B.v$	Синт
	$B.s = L.s$	Унасл
	$L.l = 1$	Синт
$B \to 0$	$B.v = 0$	Синт
$B \to 1$	$B.v = 2^{B.s}$	Синт

Пример 4. Дерево разбора и граф зависимостей для 101.01



Выводы - ?

Пример 5. Символьное дифференцирование

Атрибуты

$N.\psi$ – цепочка терминалов, выводимая из N

$N.\varphi$ – производная, представленная цепочкой терминалов

N – любой нетерминал

Грамматика:	**Семантические действия**:	
$E_0 \to E_1 + T$	$E_0.\varphi = E_1.\varphi + T.\varphi$	$E_0.\psi = E_1.\psi + T.\psi$
$E \to T$	$E.\varphi = T.\varphi$	$E.\psi = T.\psi$
$T_0 \to T_1 * P$	$T_0.\varphi = T_1.\varphi \cdot P.\psi + T_1.\psi \cdot P.\varphi$	$T_0.\psi = T_1.\psi \cdot P.\psi$
$T \to P$	$T.\varphi = P.\varphi$	$T.\psi = P.\psi$
$P \to \sin(E)$	$P.\varphi = \cos(E.\psi) \cdot E.\varphi$	$P.\psi = \sin(E.\psi)$
$P \to \cos(E)$	$P.\varphi = -\sin(E.\psi) \cdot E.\varphi$	$P.\psi = \cos(E.\psi)$
$P \to x$	$P.\varphi = 1$	$P.\psi = x$
$P \to C$	$P.\varphi = 0$	$P.\psi = C$
$P \to (E)$	$P.\varphi = E.\varphi$	$P.\psi = E.\psi$

Progr.ψ = sin(x)*x

Progr.φ = ??

Пример 5. Символьное дифференцирование (2)

- Progr.ψ = sin(x+1) * x

(sin(x+1) * x).φ = sin(x+1).φ*x.ψ + sin(x+1).ψ*x.φ=
=sin(x+1).φ*x + sin(x+1)*1=cos(x+1)$*$ (x+1).φ*x+ sin(x+1)*1=

= cos(x+1)*x $*$ (1+0) + sin(x+1)*1

S-атрибутные и L-атрибутные грамматики

Определение. Атрибутная грамматика называется S - атрибутной если все атрибуты – синтезированные.

Определение. Атрибутная грамматика называется L - атрибутной если каждый последующий атрибут символа X_j из правой части продукции

$$A \to X_1, X_2, \dots, X_n$$

зависит только от атрибутов символов $X_1, X_2, \dots, X_{j-1}$ и наследуемых атрибутов символа A

Порядок вычислений в L-атрибутной грамматике

- В L - атрибутной грамматике можно задать простой порядок вычисления атрибутов для каждого правила, а не для дерева в целом.

Рассмотрим этот порядок на примере правила

$$A \to B, C$$

1. Вычислить **унаследованные** атрибуты A
2. Вычислить унаследованные атрибуты B
3. Вычислить синтезированные атрибуты B
4. Вычислить унаследованные атрибуты C
5. Вычислить синтезированные атрибуты C
6. Вычислить **синтезированные** атрибуты A

fun -> funheader(int x) { statements; cout << x; }

Практический пример 34 + 23

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow * FT' \mid \epsilon$
$F \rightarrow i$
$F \rightarrow (E)$

F -> num
TP -> eps
T -> F TP
F -> num
TP -> eps
T -> F TP
EP -> eps
EP -> + T EP
E -> T EP

$E \Rightarrow T\ EP \Rightarrow T$+ T EP $\Rightarrow T$+ T $\epsilon \Rightarrow T$+ 23 $\Rightarrow$ F + 23 $\Rightarrow 34 + 23$

Калькулятор для нелеворекурсивной грамматики

- Рассмотрим леворекурсивную грамматику для выражений вида 2 * 3 * 4

$T \to T * F$
$T \to F$
$F \to num$

Не все парсеры умеют разбирать леворекурсивную грамматику.
Преобразуем её в эквивалентную нелеворекурсивную:

$T \to F\ T'$
$T' \to *\ F\ T'$
$T' \to \epsilon$
$F \to num$

Напишем для неё атрибутную грамматику, вычисляющую выражение. В ней будут присутствовать как синтезируемые, так и унаследованные атрибуты

Калькулятор для нелеворекурсивной грамматики (2) 2 * 3 ϵ

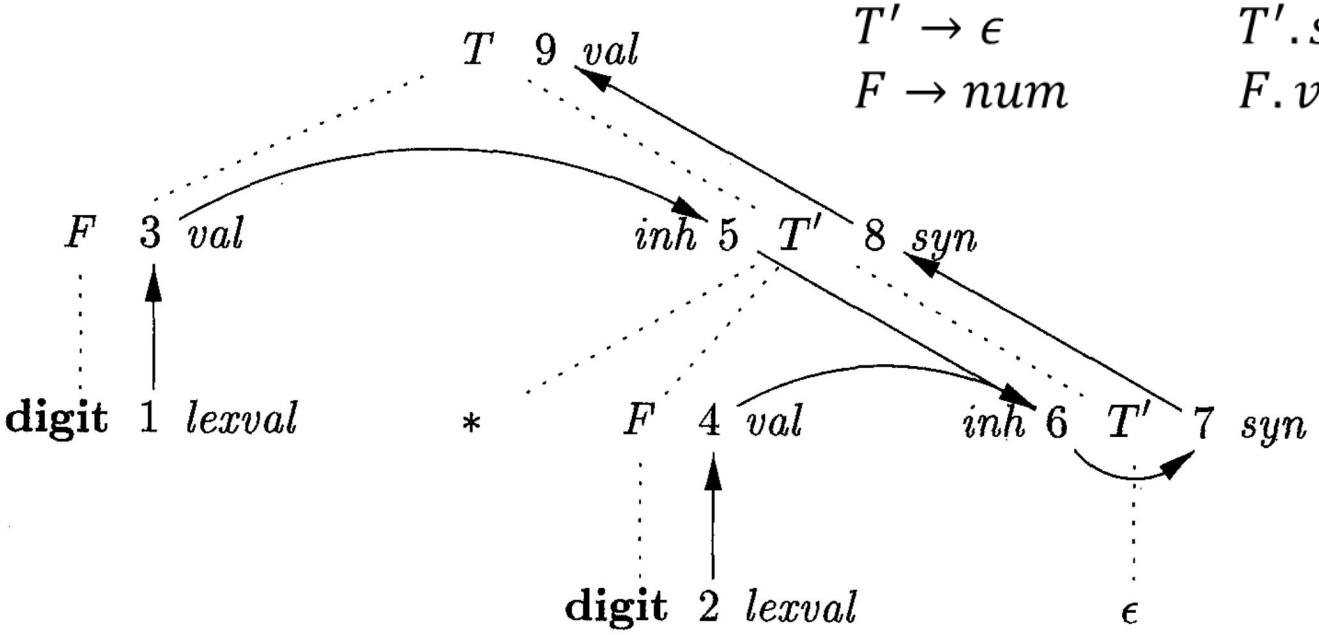
Продукция	Семантическое правило
$T \to F\ T'$	$T'.inh = F.val$
	$T.val = T'.syn$
$T' \to *\ F\ T_1'$	$T_1'.inh = T'.inh\ \times\ F.val$
	$T'.syn = T_1'.syn$
$T' \to \epsilon$	$T'.syn = T'.inh$
$F \to num$	$F.val = num.lexval$

$T \to T * F$	$T.val = T_1.val * F.val$
$T \to F$	$T.val = F.val$
$F \to num$	$F.val = num.lexval$

Калькулятор для нелеворекурсивной грамматики (3)

Граф зависимости атрибутов

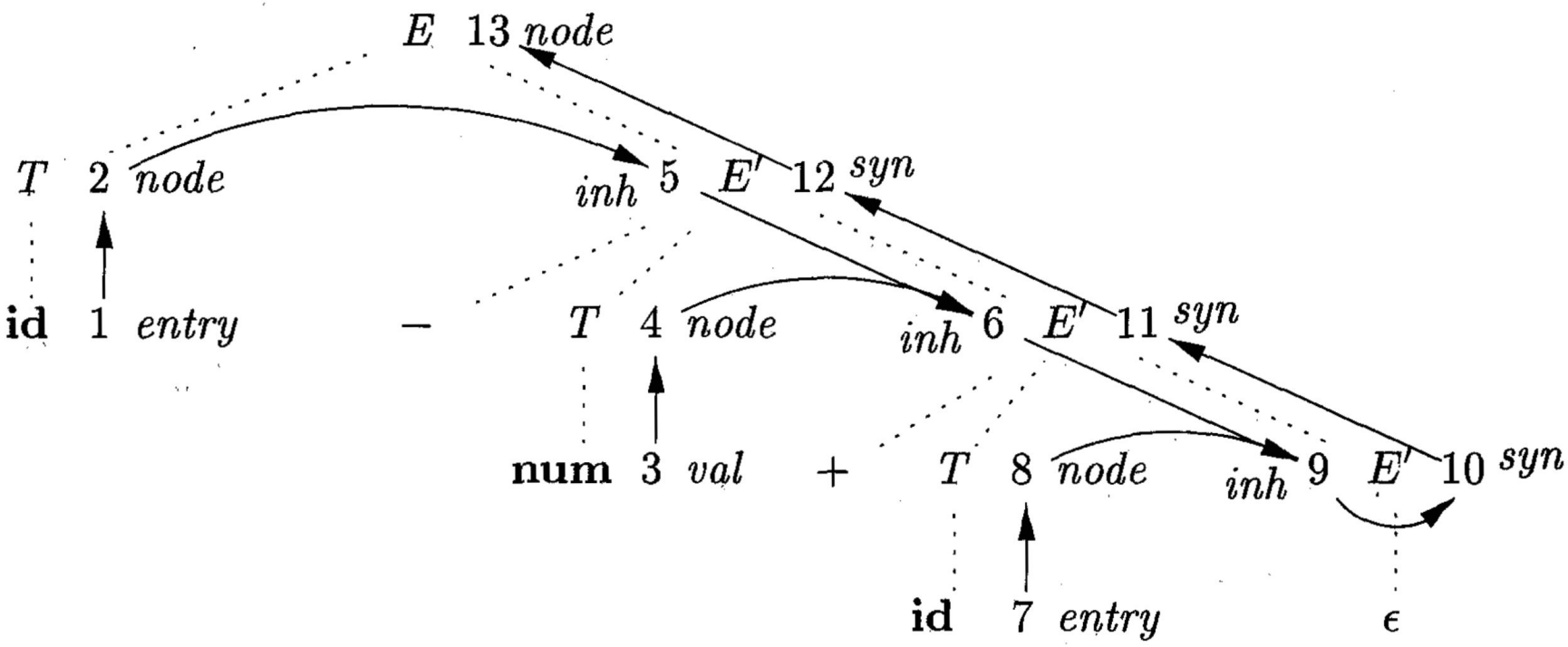
$T \to F\ T'$	$T'.inh = F.val$
	$T.val = T'.syn$
$T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$
	$T'.syn = T'_1.syn$
$T' \to \epsilon$	$T'.syn = T'.inh$
$F \to num$	$F.val = num.lexval$



Построитель AST-дерева для нелеворекурсивной грамматики 3 + 4 + 5

1)	$E \to T\ E'$	$E.node = E'.syn$ $E'.inh = T.node$
2)	$E' \to +\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3)	$E' \to -\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4)	$E' \to \epsilon$	$E'.syn = E'.inh$
5)	$T \to (\ E\ )$	$T.node = E.node$
6)	$T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
7)	$T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

Граф зависимостей для a - 4 + c



В каком порядке вычислять атрибуты

- Если граф зависимости не имеет циклов, то допустимыми порядками вычисления атрибутов $N_1, N_2, \ldots, N_n$ являются такие, что если в графе зависимостей имеется ребро от N_i к N_j, то $i < j$.

Такое упорядочение графа называется **топологической сортировкой**.

В графе без циклов всегда имеется узел, в который не входит ни одно ребро (от противного).

Сделаем этот узел первым в последовательности, удалим его из графа зависимостей и повторим процесс для оставшихся $n - 1$ узла

Q & A