



Курс «Компиляторные технологии»

## Лекция 12

# Синтаксически управляемая трансляция

**Курносов Михаил Георгиевич**

[www.mkurnosov.net](http://www.mkurnosov.net)

Сибирский государственный университет телекоммуникаций и информатики  
Весенний семестр

# Синтаксически управляемая трансляция (syntax-directed translation)

- **Синтаксически управляемое определение** (syntax-directed definition, СУО) – контекстно-свободная грамматика с *атрибутами* и *правилами*
- **Атрибуты** (attributes) связаны с символами грамматики
- **Правила** (rules) связаны с продукциями
- **Синтезируемый атрибут** нетерминала  $A$  в узле  $N$  определяется только с использованием значений атрибутов в дочерних по отношению к  $N$  узлах и в самом узле  $N$
- **Наследуемый атрибут** для нетерминала  $B$  в узле  $N$  дерева разбора определяется с использованием атрибутов в родительском по отношению к  $N$  узле, в самом узле  $N$  и дочерних узлах его родительского узла («братьях»  $N$ )
- **S-атрибутное определение** – СУО только с синтезируемыми атрибутами
- $val$  – синтезируемый атрибут

Синтаксически управляемое определение  
калькулятора (S-атрибутное)

ПРОДУКЦИЯ	СЕМАНТИЧЕСКОЕ ПРАВИЛО
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow (E)$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

# Синтаксически управляемая трансляция (syntax-directed translation)

- Весь процесс компиляции (трансляции) полностью управляется синтаксическим анализатором
- СУТ используется для построения промежуточного представления для следующей фазы компилятора: дерево разбора → промежуточное представление (например, абстрактное синтаксическое дерево)

# Синтаксически управляемое определение калькулятора (S-атрибутное)

ПРОДУКЦИЯ	СЕМАНТИЧЕСКОЕ ПРАВИЛО
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow (E)$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$



```
%{
#include <ctype.h>
%}

%token DIGIT

%%
line   : expr '\n'      { printf("%d\n", $1); }
;
expr   : expr '+' term  { $$ = $1 + $3; }
      | term
;
term   : term '*' factor { $$ = $1 * $3; }
      | factor
;
factor : '(' expr ')'   { $$ = $2; }
      | DIGIT
;

%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

- S-атрибутное СУО может быть естественным образом реализовано вместе с LR-анализатором

# Аннотированное дерево разбора

- Как строится аннотированное дерево разбора?
- В каком порядке вычисляются его атрибуты?
- Синтезируемые атрибуты можно вычислять в произвольном восходящем порядке, в обратном порядке обхода дерева разбора
- Для СУО с атрибутами обоих типов – наследуемыми и синтезируемыми – не существует гарантии наличия даже одного порядка обхода для вычисления всех атрибутов в узлах дерева разбора

ПРОДУКЦИЯ

$A \rightarrow B$

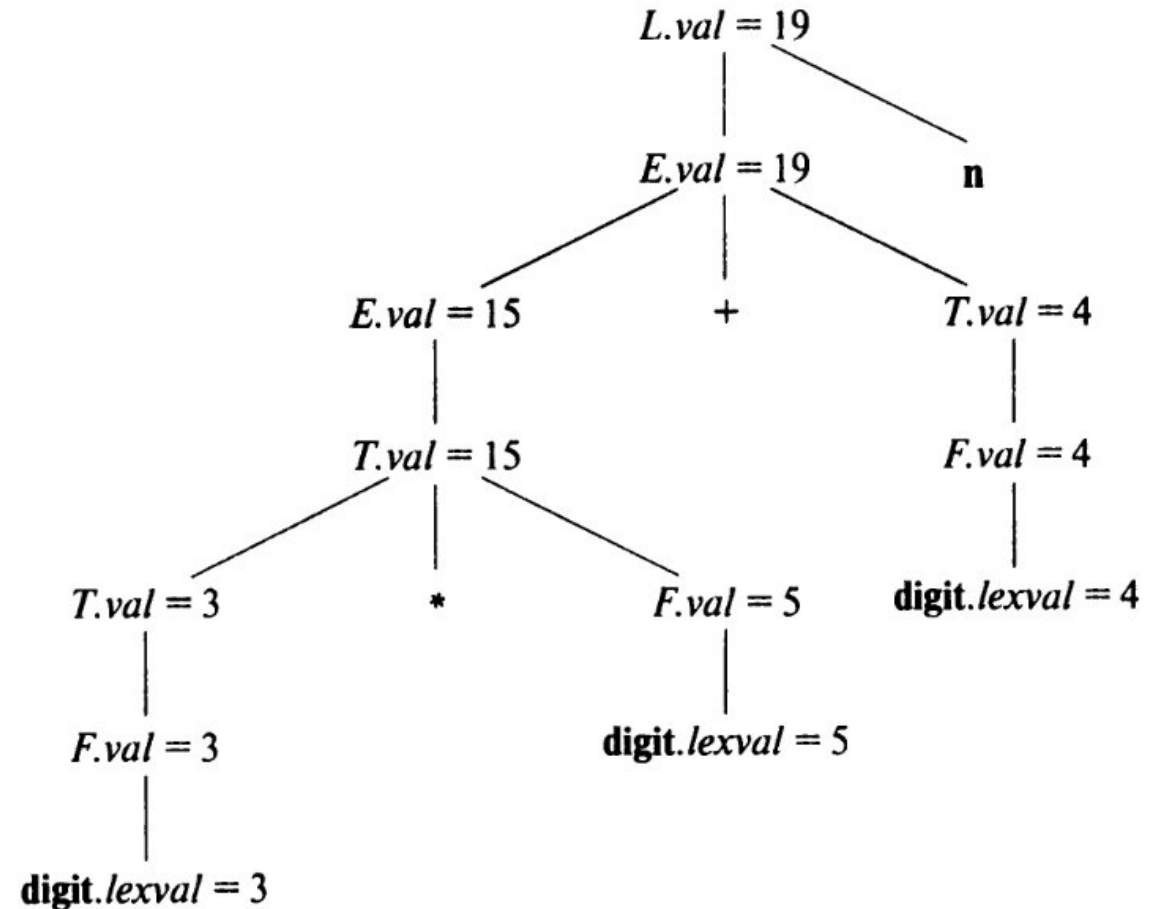
СЕМАНТИЧЕСКИЕ ПРАВИЛА

$A.s = B.i$

Цикл

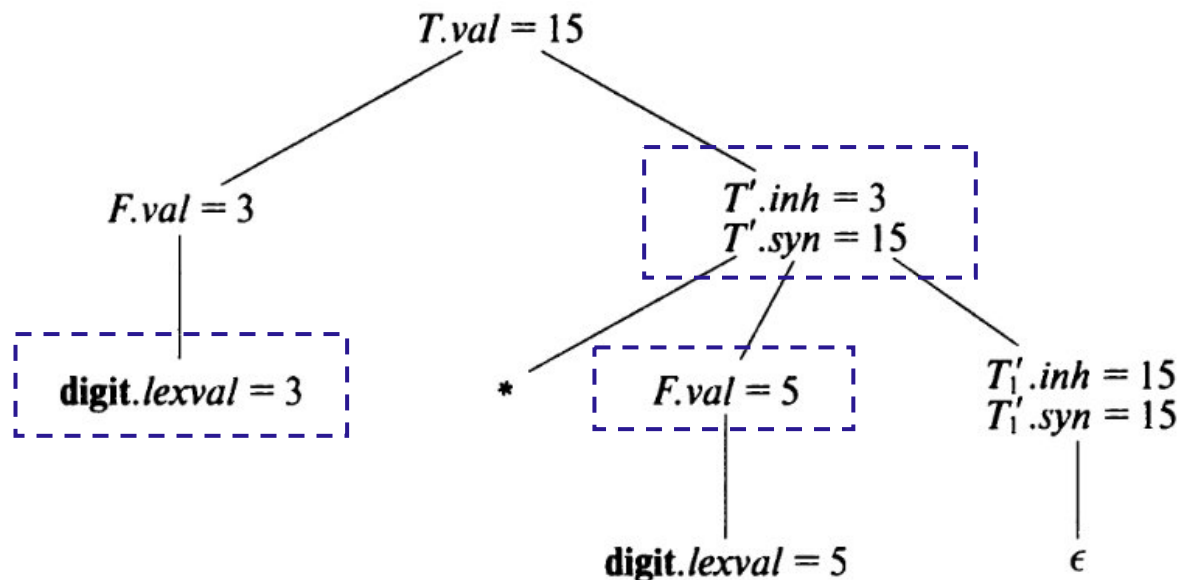
$B.i = A.s + 1$

3 \* 5 + 4 n



# Наследуемые атрибуты

- Грамматика разрабатывалась для синтаксического анализа, а не для трансляции
- Наследуемые атрибуты полезны когда структура дерева разбора «не соответствует» абстрактному синтаксису исходного кода



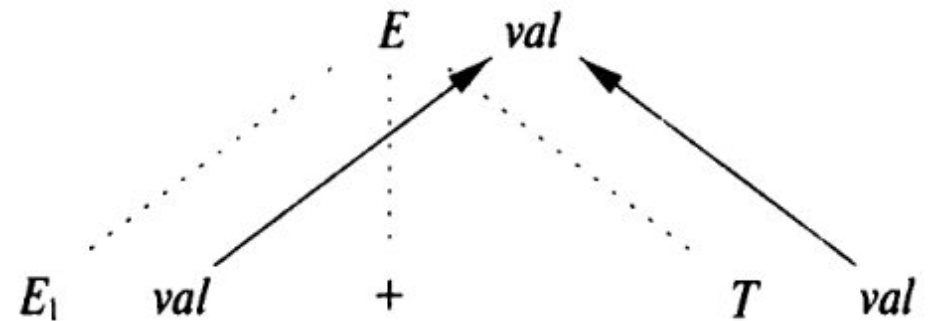
Аннотированное дерево разбора для выражение  
3 \* 5

СУО на основе грамматики для  
нисходящего синтаксического анализа

ПРОДУКЦИЯ	СЕМАНТИЧЕСКИЕ ПРАВИЛА
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow *F T_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

# Граф зависимостей

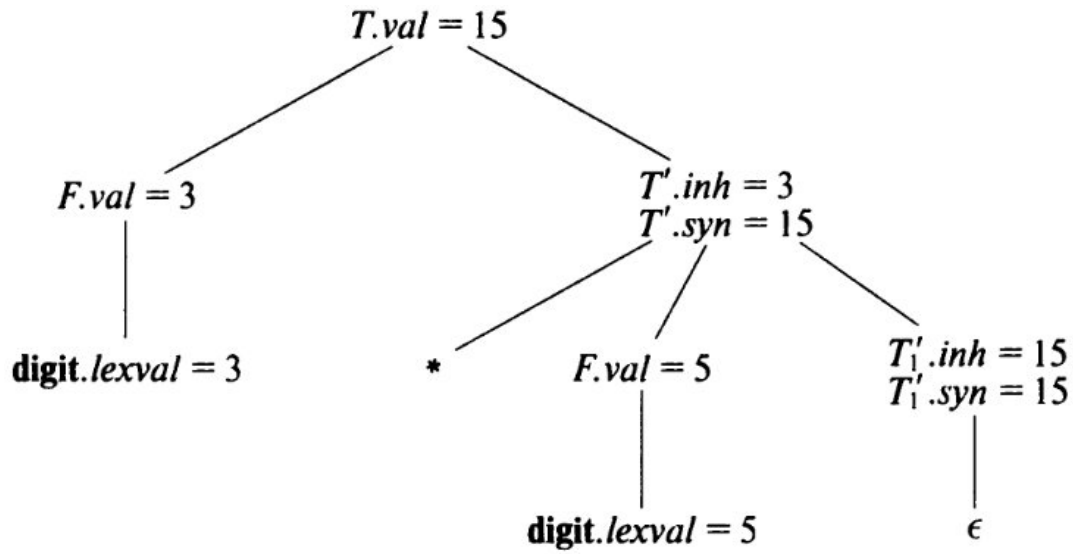
- Инструментом для установления порядка вычисления атрибутов в конкретном дереве разбора является граф зависимостей (dependency graph)
- **Граф зависимостей** (dependency graph) – поток информации между атрибутами в дереве разбора;
  - вершины графа – атрибуты
  - ребро – значение первого атрибута необходимо для вычисления второго
- Граф зависимостей определяет возможные порядки вычисления атрибутов в различных узлах дерева разбора
- Если граф зависимостей имеет ребро из узла  $M$  в узел  $N$ , то атрибут, соответствующий  $M$ , должен быть вычислен до атрибута  $N$



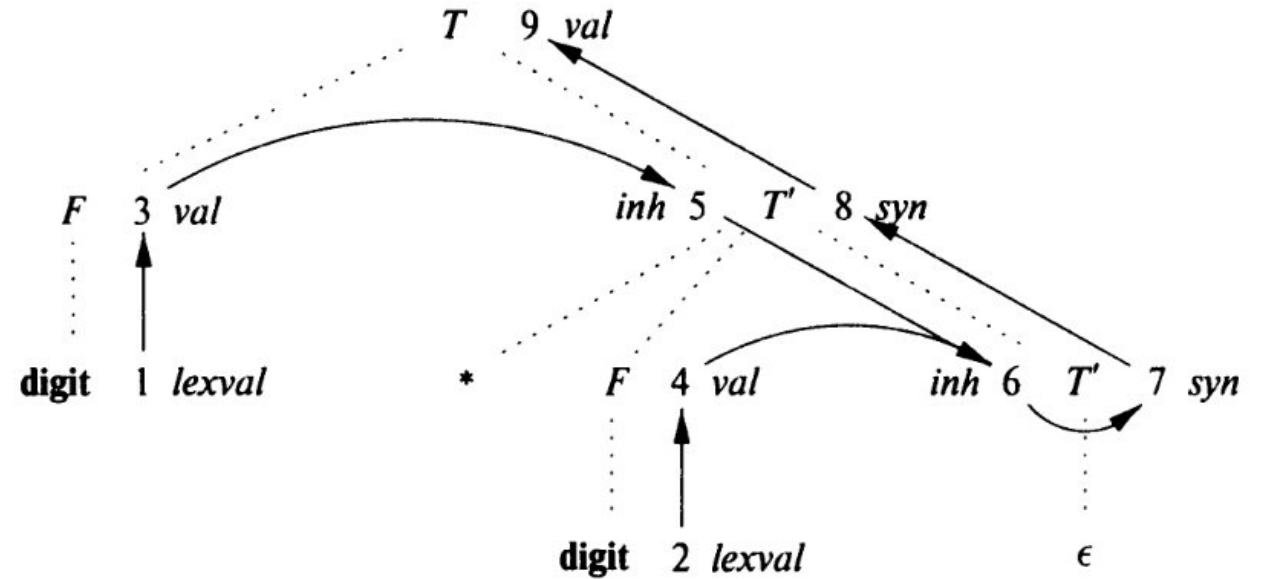
ПРОДУКЦИЯ	СЕМАНТИЧЕСКОЕ ПРАВИЛО
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$

..... ребра дерева разбора  
———— ребра графа зависимости



# Пример графа зависимостей



Аннотированное дерева разбора  
для строки 3 \* 5



Граф зависимостей для аннотированного дерева  
разбора

-  ребра дерева разбора
-  ребра графа зависимости



# Упорядочение вычисления атрибутов

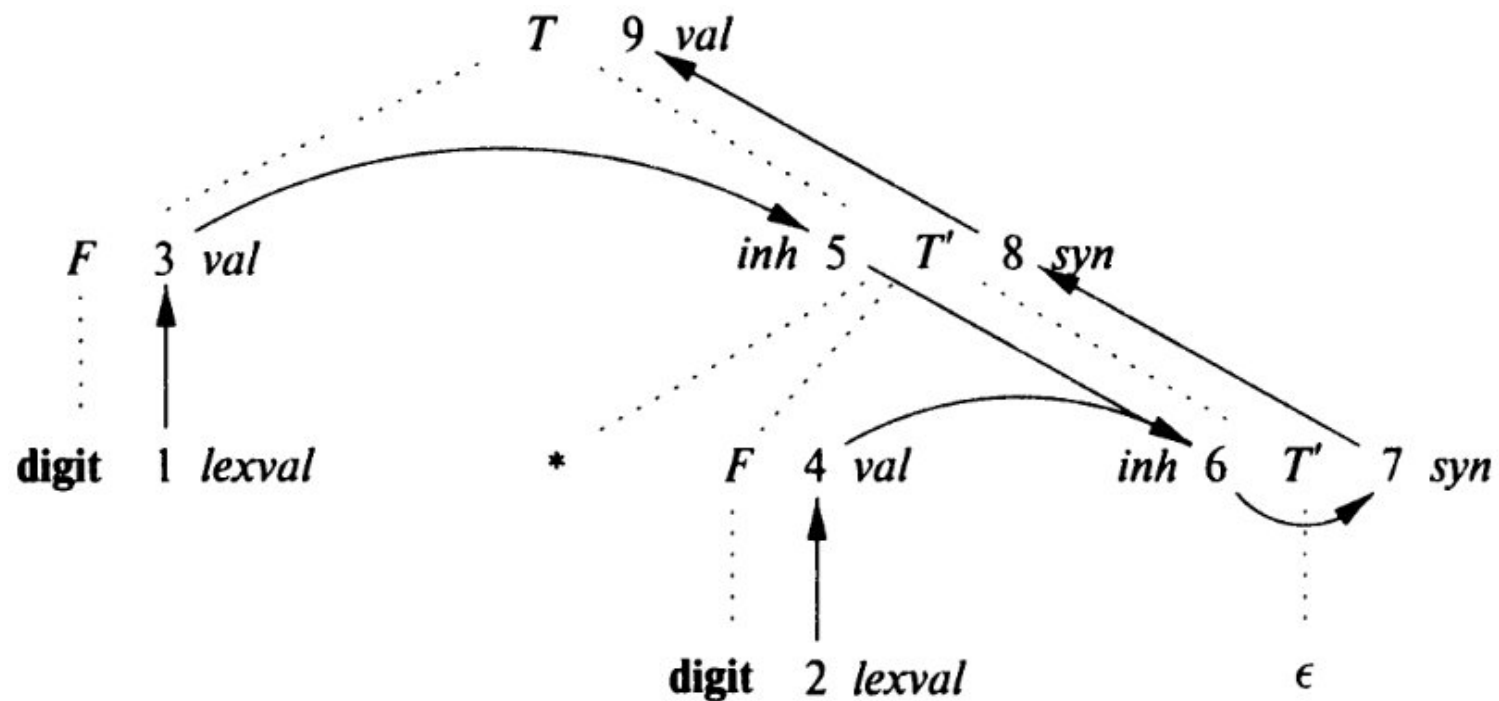
- Граф зависимостей определяет возможные порядки вычисления атрибутов в различных узлах дерева разбора
- Если граф зависимостей имеет ребро из узла  $M$  в узел  $N$ , то атрибут, соответствующий  $M$ , должен быть вычислен до атрибута  $N$
- **Допустимыми порядками** вычисления атрибутов (allowable orders) являются последовательности узлов

$$N_1, N_2, \dots, N_k$$

такие, что имеется ребро от  $N_i$  к  $N_j$ , то  $i < j$

- Упорядочение графа зависимостей, которое выстраивает узлы в линейном порядке называется **топологической сортировкой** (topological sort)
- Если в графе имеется цикл, топологическая сортировка графа невозможна
- Если в графе зависимостей дерева разбора присутствует цикл значит невозможно и вычисление СУО для данного дерева разбора

# Топологические сортировки



- 1, 2, 3, 4, 5, 6, 7, 8, 9
- 1, 3, 5, 2, 4, 6, 7, 8, 9

# S-атрибутивные определения

- **Синтаксически управляемое определение** является **S-атрибутивным**, если все его атрибуты синтезируемые
- Атрибуты S-атрибутивного СУО могут вычисляться в любом восходящем порядке узлов в дереве разбора
- Путем обхода дерева разбора в обратном порядке (postorder)

```
postorder (N) {  
    for ( каждый дочерний узел C узла N, начиная слева) postorder (C);  
    Вычислить атрибуты, связанные с узлом N;  
}
```

- S-атрибутивные определения могут быть реализованы во время нисходящего синтаксического анализа, он соответствует обходу в обратном порядке
- Обратный порядок обхода соответствует порядку, в котором LR-синтаксический анализатор сворачивает тела productions в их заголовки

# L-атрибутные определения

- Ребра графа зависимостей между атрибутами, связанными с телом продукции, идут только слева направо, но не справа налево
- Каждый атрибут должен быть:
  - синтезируемым
  - наследуемым, но при выполнении ограничений:  
имеется продукция  $A \rightarrow X_1X_2...X_n$  и существует наследуемый атрибут  $X_i.a$ , вычисляемый при помощи правила, связанного с данной продукцией, это правило может использовать только:
    - наследуемые атрибуты, связанные с заголовком  $A$
    - наследуемые либо синтезируемые атрибуты, связанные с вхождениями символов  $X_1X_2...X_{i-1}$ , расположенных слева от  $X_i$
    - наследуемые либо синтезируемые атрибуты, связанные с вхождениями самого  $X_i$  но только таким образом, что в графе зависимостей, образованном атрибутами этого  $X_i$  нет циклов

L-атрибутное СУО	Продукция	СЕМАНТИЧЕСКОЕ ПРАВИЛО
	$T \rightarrow F T'$	$T'.inh = F.val$
	$T' \rightarrow *F T'_1$	$T'_1.inh = T'.inh \times F.val$

# Применения синтаксически управляемой трансляции

- Проверка типов данных
- Генерация промежуточного кода
- Построение синтаксических деревьев – узлы дерева генерируются в правилах СУО

# Построение синтаксических деревьев

- Узел абстрактного синтаксического дерева – языковая конструкция, дочерние узлы представляют значащие компоненты этой конструкции
- Узлы строятся на базе объектов
- $Leaf(op, val)$  – лист, с лексическим значением в  $val$
- $Node(op, c_1, c_2, \dots, c_k)$  – внутренний узел АСТ с  $k$  дочерними узлами

S-атрибутное определение строит синтаксическое дерево для грамматики выражений с операторами: + и – (нетерминалы имеют один синтезируемый атрибут  $node$  – узел синтаксического дерева)

ПРОДУКЦИЯ	СЕМАНТИЧЕСКОЕ ПРАВИЛО
1) $E \rightarrow E_1 + T$	$E.node = \mathbf{new Node} (' + ', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \mathbf{new Node} (' - ', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
4) $T \rightarrow (E)$	$T.node = E.node$
5) $T \rightarrow \mathbf{id}$	$T.node = \mathbf{new Leaf}(\mathbf{id}, \mathbf{id.entry})$
6) $T \rightarrow \mathbf{num}$	$T.node = \mathbf{new Leaf}(\mathbf{num}, \mathbf{num.val})$

# Построение синтаксических деревьев

S-атрибутное определение

ПРОДУКЦИЯ	СЕМАНТИЧЕСКОЕ ПРАВИЛО
1) $E \rightarrow E_1 + T$	$E.node = \mathbf{new Node} (' + ', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \mathbf{new Node} (' - ', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
4) $T \rightarrow (E)$	$T.node = E.node$
5) $T \rightarrow \mathbf{id}$	$T.node = \mathbf{new Leaf}(\mathbf{id}, \mathbf{id.entry})$
6) $T \rightarrow \mathbf{num}$	$T.node = \mathbf{new Leaf}(\mathbf{num}, \mathbf{num.val})$

Шаги построения синтаксического  
дерева для выражения:  $a - 4 + c$

(правила вычисляются в порядке обратного  
обхода дерева разбора)

$p_1 = \mathbf{new Leaf}(\mathbf{id}, \mathbf{entry-a})$ ;

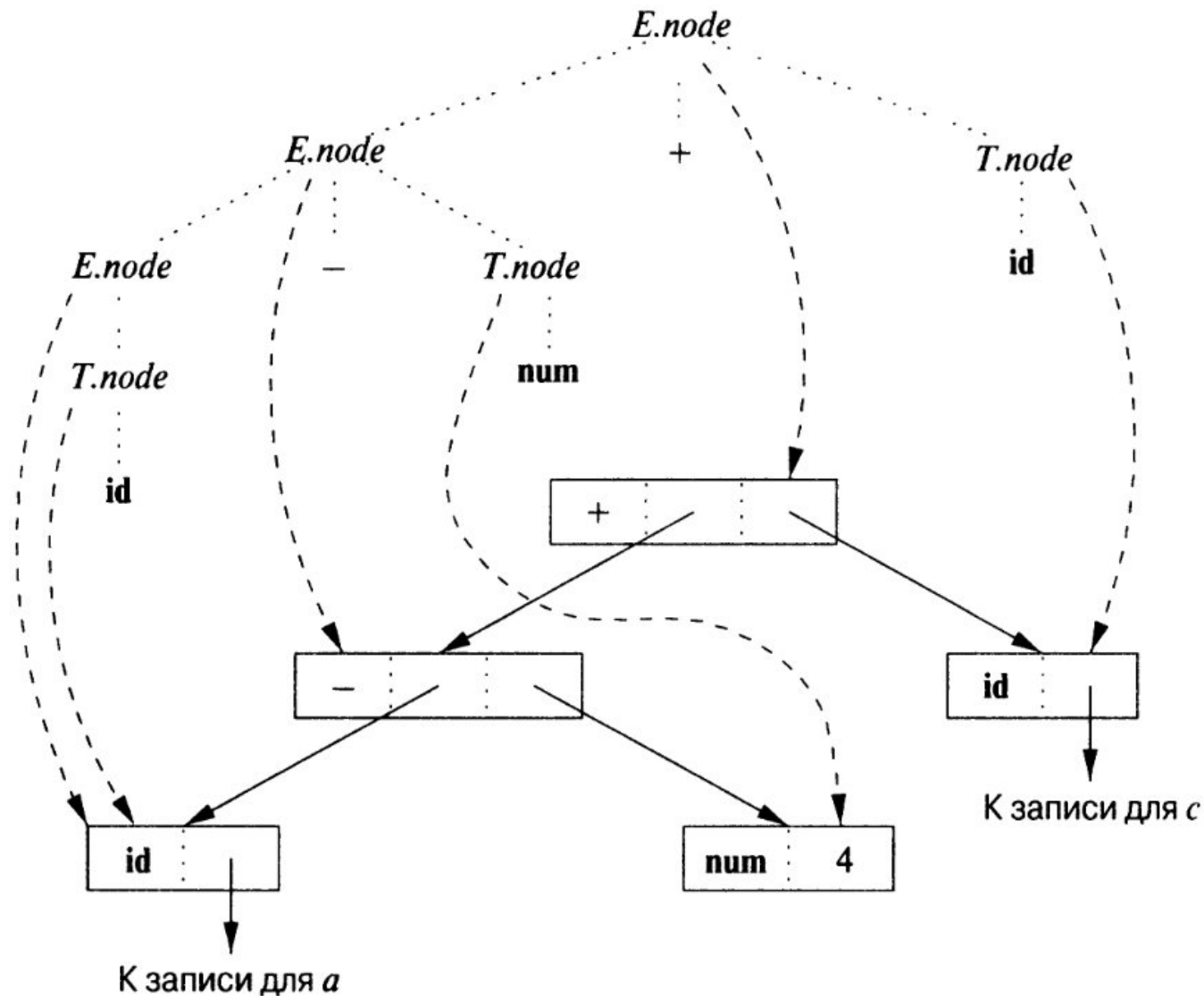
$p_2 = \mathbf{new Leaf}(\mathbf{num}, 4)$ ;

$p_3 = \mathbf{new Node} (' - ', p_1, p_2)$ ;

$p_4 = \mathbf{new Leaf}(\mathbf{id}, \mathbf{entry-c})$ ;

$p_5 = \mathbf{new Node} (' + ', p_3, p_4)$ ;

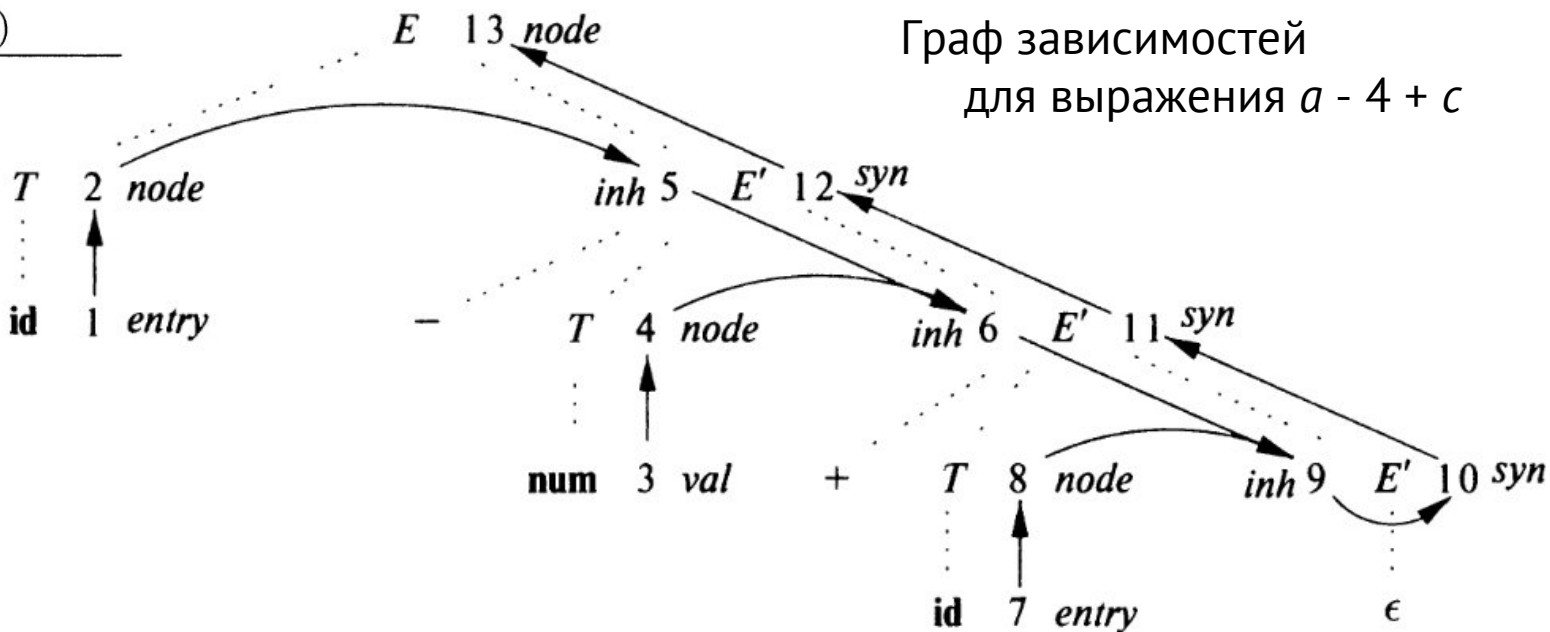
- $p_5$  – корень построенного АСТ



# Построение синтаксических деревьев

## L-атрибутное определение

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

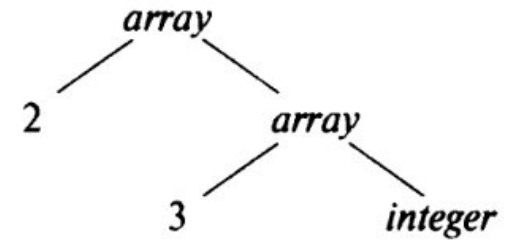




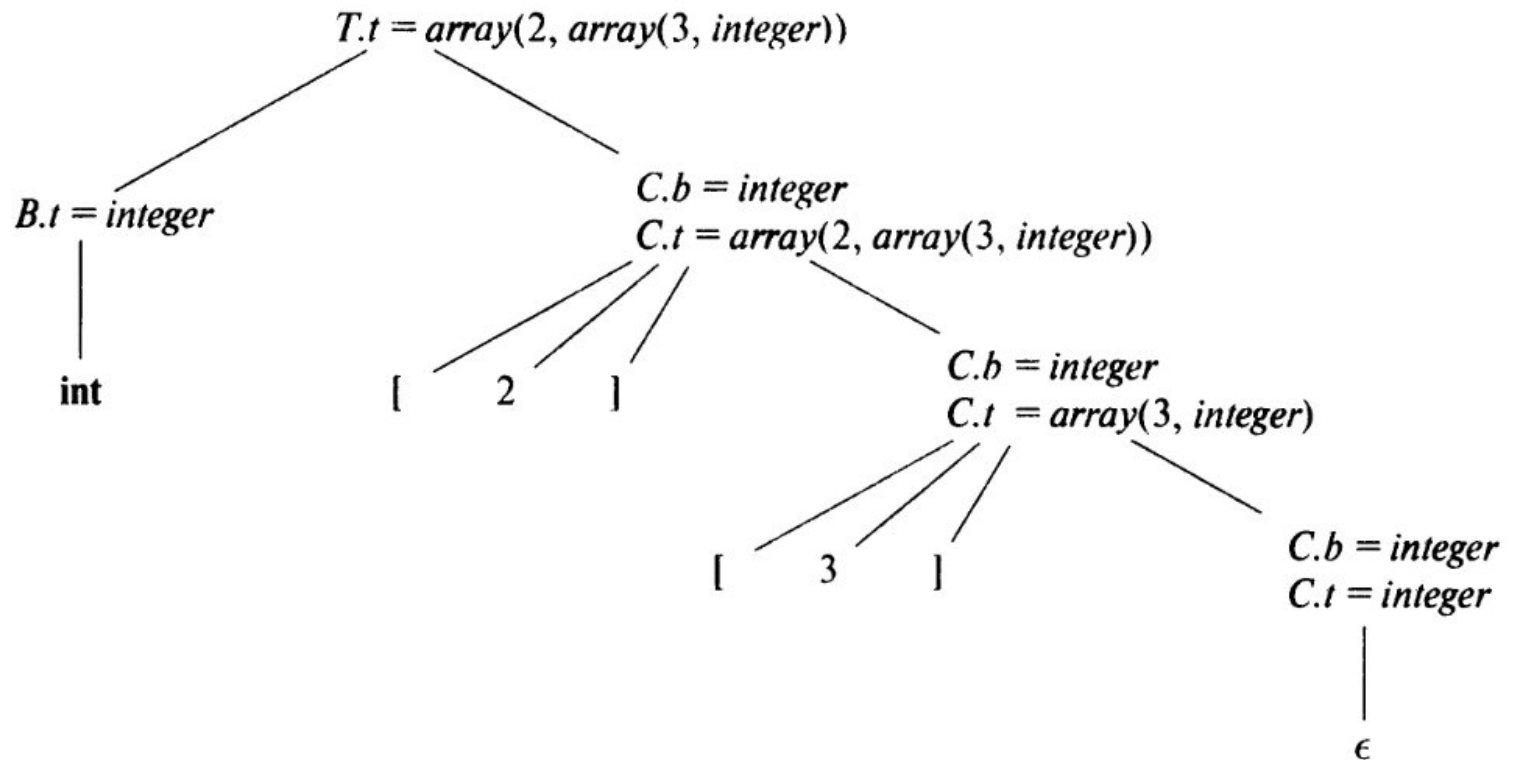
# Синтаксически управляемая трансляция типов массивов

- Массив из двух массивов по 3 целых числа  
`int [2][3]`

`array(2, array(3, int))`



ПРОДУКЦИЯ	СЕМАНТИЧЕСКИЕ ПРАВИЛА
$T \rightarrow B C$	$T.t = C.t$ $C.b = B.t$
$B \rightarrow \mathbf{int}$	$B.t = \mathit{integer}$
$B \rightarrow \mathbf{float}$	$B.t = \mathit{float}$
$C \rightarrow [\mathbf{num}] C_1$	$C.t = \mathit{array}(\mathbf{num.val}, C_1.t)$ $C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$



# Синтаксически управляемые схемы трансляции

- **Синтаксически управляемая схема трансляции** (СУТ, syntax-directed translation scheme, SDT) – контекстно-свободная грамматика с программными фрагментами, внедренными в тела продукций (семантические действия)
- Любая СУТ может быть реализована путем построения дерева разбора с последующим выполнением действий в порядке в глубину слева направо – в порядке прямого обхода дерева (preorder)
- Обычно СУТ реализуется в процессе синтаксического анализа, без построения дерева разбора
- Двух важных случая:
  1. Грамматика поддается LR-синтаксическому анализу, СУО – S-атрибутное
  2. Грамматика поддается LL-синтаксическому анализу, СУО – L-атрибутное

# Постфиксные схемы трансляции

- Простейшая реализация СУТ имеет место в случае восходящего синтаксического анализа и S-атрибутного СУО
- Можно построить СУТ, в которой каждое действие размещается в конце продукции и выполняется вместе со сверткой тела продукции в заголовок
- **Постфиксная СУТ** (postfix SDT) – СУТ со всеми действиями, расположенными на правом конце тел продукций

- Грамматика – LR-грамматика
- СУО – S-атрибутное

$$\begin{aligned} L &\rightarrow E \mathbf{n} && \{ \mathit{print}(E.val); \} \\ E &\rightarrow E_1 + T && \{ E.val = E_1.val + T.val; \} \\ E &\rightarrow T && \{ E.val = T.val; \} \\ T &\rightarrow T_1 * F && \{ T.val = T_1.val \times F.val; \} \\ T &\rightarrow F && \{ T.val = F.val; \} \\ F &\rightarrow (E) && \{ F.val = E.val; \} \\ F &\rightarrow \mathbf{digit} && \{ F.val = \mathbf{digit.lexval}; \} \end{aligned}$$

- Атрибут каждого грамматического символа может помещаться в стек в том месте, где он может быть обнаружен в процессе свертки
- Лучше разместить атрибуты вместе с грамматическими символами (или LR-состояниями, представляющими эти символы) в записях стека.

Постфиксная СУТ

# СУТ для L-атрибутивных определений

- Более общий случай – L-атрибутивное СУО
- Пусть грамматика поддается нисходящему синтаксическому анализу (top-down)
- Присоединим к дереву разбора семантические действия и выполним их в процессе обхода дерева в прямом порядке
- **Алгоритм перехода от L-атрибутивного СУО к СУТ**
- Вставить действие, которое вычисляет наследуемые атрибуты нетерминала  $A$ , непосредственно перед вхождением  $A$  в тело продукции (если несколько наследуемых атрибутов ациклически зависят друг от друга, следует упорядочить вычисление атрибутов)
- Поместить действия, вычисляющие синтезируемые атрибуты заголовка продукции, в конце тела соответствующей продукции

# Реализация L-атрибутных СУО: синтаксический анализ методом рекурсивного спуска

- Для каждого нетерминала  $A$  анализатор имеет отдельную функцию  $A$
- Расширим синтаксический анализатор с учетом L-атрибутного СУО:
  1. Аргументами функции  $A$  являются наследуемые атрибуты нетерминала  $A$
  2. Возвращаемое функцией  $A$  значение представляет собой набор синтезируемых атрибутов нетерминала  $A$

## Реализация конструкции **while**

```
string S(label next) {  
    string Scode, Ccode; /* Локальные переменные с фрагментами кода */  
    label L1, L2; /* Локальные метки */  
    if ( Текущий входной символ == токен while ) {  
        Перемещение по входному потоку;  
        Проверить наличие '(' во входной строке и перейти к новой  
            позиции;  
        L1 = new();  
        L2 = new();  
        Ccode = C(next, L2);  
        Проверить наличие ')' во входной строке и перейти к новой  
            позиции;  
        Scode = S(L1);  
        return("label1"||L1||Ccode||"label1"||L2||Scode);  
    }  
    else /* Инструкции других видов */  
}
```

# Реализация L-атрибутивных СУО: генерация кода «на лету» (on-the-fly)

- Инкрементно генерируются части кода с записью в массив или выходной файл при помощи действий из СУТ

## Реализация конструкции **while**

```
void S(label next) {  
    label L1, L2; /* Локальные метки */  
    if ( Текущий символ == токен while ) {  
        Перемещение по входному потоку;  
        Проверить наличие '(' во входной строке и перейти к новой  
            позиции;  
        L1 = new();  
        L2 = new();  
        print("label ", L1);  
        C(next, L2);  
        Проверить наличие ')' во входной строке и перейти к новой  
            позиции;  
        print("label ", L2);  
        S(L1);  
    }  
    else /* Инструкции других видов */  
}
```