Лекция 7

15 октября

6. ТРАНСЛИРУЮЩИЕ ГРАММАТИКИ

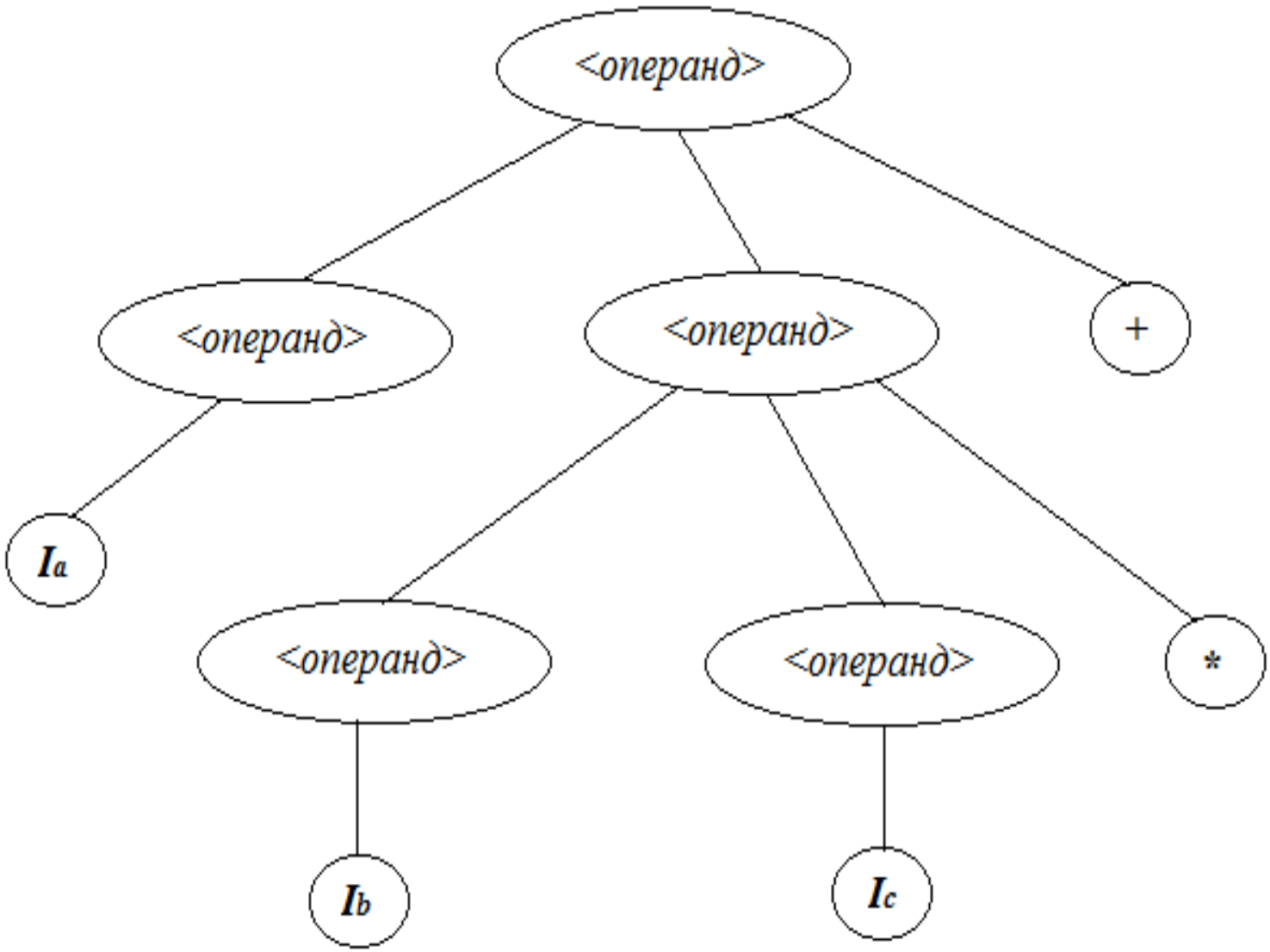
6.1. Цепочечный перевод

- Рассмотрим трансляцию на примере перевода арифметических выражений из одной формы записи в другую.
- Обычный способ записи арифметических выражений называется *инфиксная запись.* В этом случае знаки операций записываются между операндами. Существуют и другие способы записи арифметических выражений, в частности *постфиксная запись* или *обратная польская запись*.
- При использовании постфиксной записи знаки операций записываются после операндов:
-
- *Инфиксная запись* *Постфиксная запись*
- *a + b * c* *a b c * +*
-
- Выражение в постфиксной форме вычисляется следующим образом. Мы движемся по цепочке, состоящей из операндов и знаков операций, слева направо. Как только встречается знак операции, берутся два предшествующих ему операнда, выполняется операция, и результат помещается на место этих операндов и знака операции. Далее продолжается движение слева направо, и выполняются аналогичные действия. Ещё одной особенностью постфиксной записи является то, что здесь не требуются скобки для задания приоритета операций.

Грамматика польских выражений

- Множество польских выражений с операциями + и * можно породить с помощью следующей грамматики:
-
- ***<операнд> → <операнд> <операнд> +***
- ***<операнд> → <операнд> <операнд> ****
- ***<операнд> → I***
-
- Для приведённого примера дерево вывода имеет следующий вид:

Синтаксическое дерево



Механизм перевода инфиксной записи в постфиксную

- Рассмотрим теперь, как мог бы работать некоторый механизм, осуществляя перевод инфиксной записи в постфиксную. Для нашего выражения последовательность действий будет следующая:
-
- *Читать(a) Печатать(a) Читать(+) Читать(b) Печатать(b) Читать(*)*
- *Читать(c) Печатать(c) Печатать(*) Печатать(+)*
-
-
- Таким образом, последовательность действий ввода и вывода для нашего выражения будет выглядеть следующим образом:
-
- a {a} + b {b} * c {c} {*} {+}
-
- Если теперь выбрать только действия вывода, то получим
-
- {a} {b} {c} {*} {+}

Исходная грамматика

-
-
- А после выполнения самих действий вывода получаем постфиксную запись
-
- a b c * +
-
-
- Вернёмся к грамматике для арифметических (инфиксных) выражений:
-
- ***<E> → <E> + <T>***
- ***<E> → <T>***
- ***<T> → <T> * <P>***
- ***<T> → <P>***
- ***<P> → (<E>)***
- ***<P> → I***
-
-

Модифицированная грамматика

- *<P> → I { I }*
- *<E> → <E> + <E> {+}*
- *<E> →<E> + <T> {+}*
- *<E> →<T>*
- *<T> → <T> * <P> {*}*
- *<T> → <P>*
- *<P> → (<E>)*
- *<P> → I {I}*

Вывод цепочки

- Полученная грамматика представляет собой то, что обычно называют *транслирующей грамматикой* или *грамматикой перевода*. Поскольку действия, заключённые в скобки, в данном случае являются выдачей символа, такие грамматики часто называют *грамматиками цепочечного перевода*.
- *Транслирующей грамматикой* или *грамматикой перевода* называется контестно-свободная грамматика, множество терминальных символов которой разбито на множество входных символов и множество символов действия.
- Цепочки языка, порождаемого транслирующей грамматикой, представляют собой последовательности символов действия.
- Рассмотрим, как осуществляется перевод выражения ***a + b * c***.
- Вывод этого выражения по исходной грамматике выглядит следующим образом:
-
- ***<E> => <E> + <T> => <T> + <T> =>* <P> + <T> * <P> =>***
- ***=> <P> + <P> * <P> =>* $l_a + l_b * l_c$***
-
- т.е. ***a + b * c***.

Вывод на основе транслирующей грамматики

- В случае транслирующей грамматики получаем:
-
- ***<E> => <E> + <T> {+} => <T> + <T> {+} => <T> + <T> * <P> {*} {+} =>***
- ***=> <P> + <T> * <P> {*} {+} => <P> + <P> * <P> {*} {+} =>***
- ***=> I_a {I_a} + <P> * <P> {*} {+} => I_a {I_a}+ I_b {I_b} * <P> {*} {+} =>***
- ***=> I_a {I_a}+ I_b {I_b} * I_c {I_c} {*} {+}***
-
- Оставляя только символы действия, получаем:
-
- ***{I_a} {I_b} {I_c} {*} {+}***
-
- Теперь, выполнив эти действия, получим постфиксную форму ***abc*+***.

L L (1)-грамматика

- Рассмотрим другую исходную грамматику для арифметических выражений:
-
- ***1. <E> → <T> <E-список>***
- ***2. <E-список> → + <T> <E-список>***
- ***3. <E-список> → ε***
- ***4. <T> → <P> <T-список>***
- ***5. <T-список> → * <P> <T-список>***
- ***6. <T-список> → ε***
- ***7. <P> → (<E>)***
- ***8. <P> → I***
- ***9. <P> → C***
-
-
-

Транслирующая L L (1)-грамматика

-
-
- Следовательно, в правиле 2 действие ***{+}*** необходимо поместить после нетерминала ***<T>***, поскольку именно этот нетерминал является вторым операндом операции сложения. В итоге получаем грамматику:
-
- ***1. <E> → <T> <E-список>***
- ***2. <E-список> → + <T> {+} <E-список>***
- ***3. <E-список> → ε***
- ***4. <T> → <P> <T-список>***
- ***5. <T-список> → * <P> {*} <T-список>***
- ***6. <T-список> → ε***
- ***7. <P> → (<E>)***
- ***8. <P> → I {I}***
- ***9. <P> → C {C}***

Управляющая таблица

Магазинные символы	I	C	+	*	(	)	⊥
<E>	#1	#1	Отверг	Отверг	#1	Отверг	Отверг
<T>	#4	#4	Отверг	Отверг	#4	Отверг	Отверг
<E-сп>	Отверг	Отверг	#2	Отверг	Отверг	#3	#3
<P>	#8	#9	Отверг	Отверг	#7	Отверг	Отверг
<T-сп>	Отверг	Отверг	#6	#5	Отверг	#6	#6
)	Отверг	Отверг	Отверг	Отверг	Отверг	Вытолк Сдвиг	Отверг
∇	Отверг	Отверг	Отверг	Отверг	Отверг	Отверг	Допустить
{+}	Выдать(+), Вытолкнуть, Держать						
{*}	Выдать(*), Вытолкнуть, Держать						
Начальное содержимое магазина <E>∇							

Элементы таблицы

- #1 *Заменить(**<T> <E-список>**), Держать*
- #2 *Заменить(**<T> {+} <E-список>**), Сдвиг*
- #3 *Вытолкнуть, Держать*
- #4 *Заменить(**<P> <T-список>**), Держать*
- #5 *Заменить(**<P> {*} <T-список>**), Сдвиг*
- #6 *Вытолкнуть, Держать*
- #7 *Заменить(**<E>)**), Сдвиг*
- #8 *Выдать(I), Вытолкнуть, Сдвиг*
- #9 *Выдать(C),Вытолкнуть, Сдвиг*
-
-

6.2. Атрибутные транслирующие грамматики

- Необходимо расширить понятие транслирующей грамматики, чтобы включить в перевод и значение символа.
- Такая расширенная грамматика называется *атрибутной грамматикой*.
- Главная идея, лежащая в основе атрибутной грамматики, состоит в том, что значения символов сопоставляются всем вершинам дерева вывода, как терминальным, так и не терминальным.

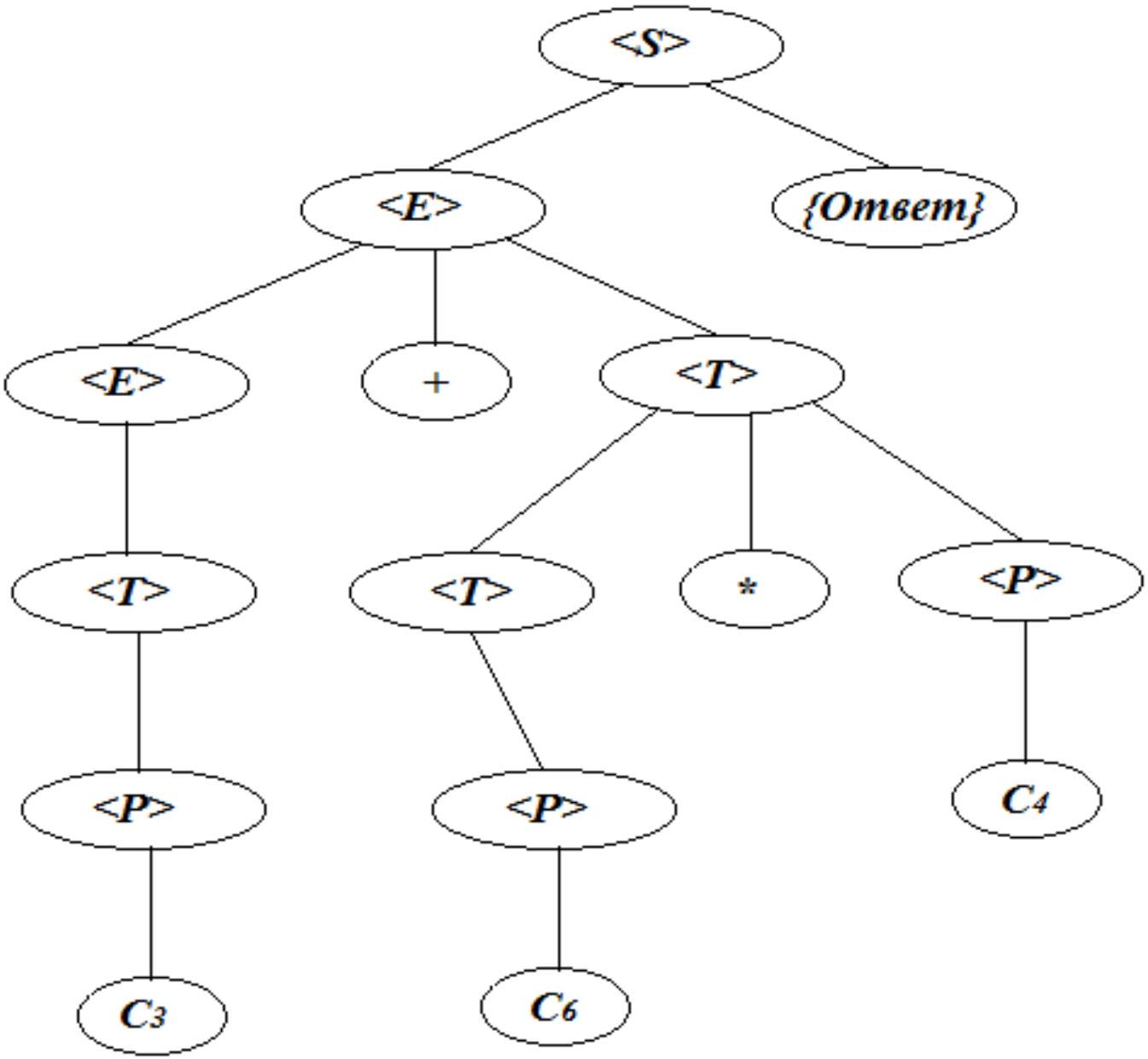
Пример

- Грамматика цепочечного перевода в данном случае будет выглядеть следующим образом:
-
- ***1. <S> → <E> {Ответ }***
- ***2. <E> → <E> + <T>***
- ***3. <E> → <T>***
- ***4. <T> → <T> * <P>***
- ***5. <T> → <P>***
- ***6. <P> → (<E>)***
- ***7. <P> → C***
-

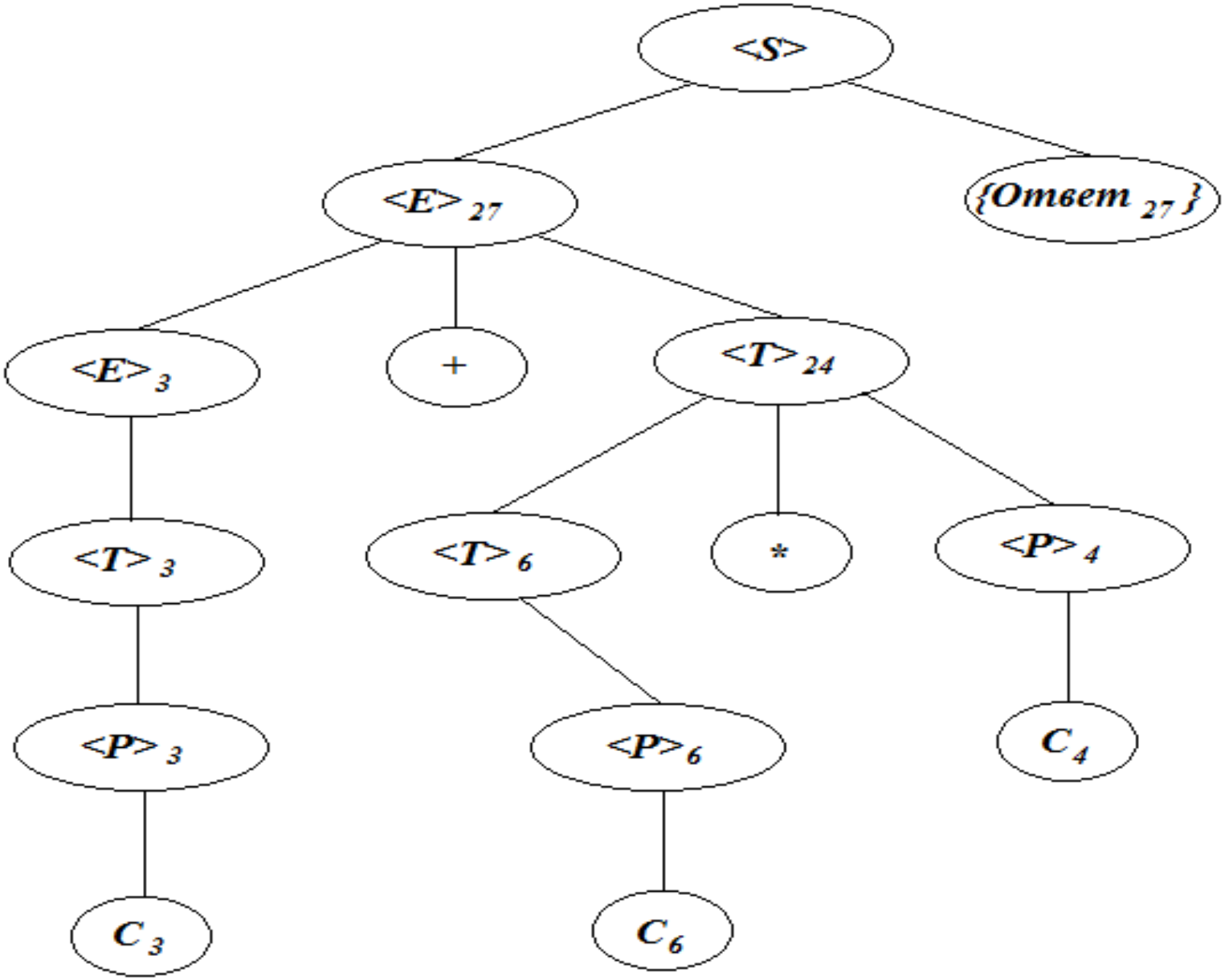
Пример (продолжение)

- Значение выходного символа ***Ответ*** должно быть числом, равным значению исходного арифметического выражения.
- Рассмотрим теперь конкретную входную цепочку
-
- $C_3 + C_6 * C_4$,
-
- где значения входных лексем, выданных лексическим блоком, указаны индексами.
- Дерево вывода для данной входной цепочки имеет следующий вид:

Синтаксическое дерево



Атрибутное синтаксическое дерево



Правила вычисления атрибутов

- Мы сопоставили нетерминалам атрибуты, добавим теперь правила вычисления атрибутов.
- Например, для правила 2 получим:
-
- $\langle E \rangle_p \rightarrow \langle E \rangle_q + \langle T \rangle_r$
- $p := q + r$
-
- Атрибуты такого рода обычно называют *ситезируемыми* атрибутами, т.к. атрибут некоторого узла дерева вычисляется на основе атрибутов его потомков.
- В итоге получаем следующую атрибутную грамматику:

Атрибутная транслирующая грамматика

- *1. <S> → $<E>_p$ {Ответ $_q$ }*
- *p := q*
- *2. $<E>_p$ → $<E>_q$ + $<T>_r$*
- *p := q + r*
- *3. $<E>_p$ → $<T>_q$*
- *p :=q*
- *4. $<T>_p$ → $<T>_q$ * $<P>_r$*
- *p := q + r*
- *5. $<T>_p$ → $<P>_q$*
- *p := q*
- *6. $<P>_p$ → (<E>$_q$)*
- *p := q*
- *7. $<P>_p$ → C_q*
- *p := q*

Другая грамматика

- Рассмотрим теперь другую ситуацию.
- Пусть дана следующая грамматика с начальным символом *<описание>*:
-
- ***1. <описание> → <тип> I <список переменных>;***
- ***2. <список переменных> → , I <список переменных>***
- ***3. <список переменных> → ε***
- ***4. <тип> → int***
- ***5. <тип> → real***
- ***6. <тип> → bool***

Переход к транслирующей грамматике

- При обработке описания синтаксический блок для каждой переменной вызывает процедуру
-
- *Установить_тип*,
-
- которая помещает один из типов ***int, real*** или ***bool*** в надлежащее поле элемента таблицы идентификаторов, соответствующего данной переменной.
-
-
-
-

Транслирующая грамматика

- В итоге получаем следующую грамматику:
-
- ***1. <описание> → <тип> I {Установить_тип} <список***
- ***переменн>;***
- ***2. <список переменн> → , I {Установить_тип} <список***
- ***переменн>***
- ***3. <список переменн> → ε***
- ***4. <тип> → int***
- ***5. <тип> → real***
- ***6. <тип> → bool***

Добавление атрибутов

- Процедура *Установить_тип* должна иметь два аргумента: указатель на элемент таблицы идентификаторов, соответствующий переменной, и тип переменной.
- Тогда вызов процедуры *Установить_тип* можно записать следующим образом:
-
- *Установить_тип (указатель, тип)*
-
- Нам необходимо ввести в нашу грамматику атрибуты и правила их выполнения.
- Тогда вхождения символа действия *Установить_тип* будут иметь следующий вид:
-
- { *Установить_тип* $_{p,t}$ }
-
- где ***p*** – указатель, а ***t*** – значение, характеризующее тип.

Добавление атрибутов (продолжение)

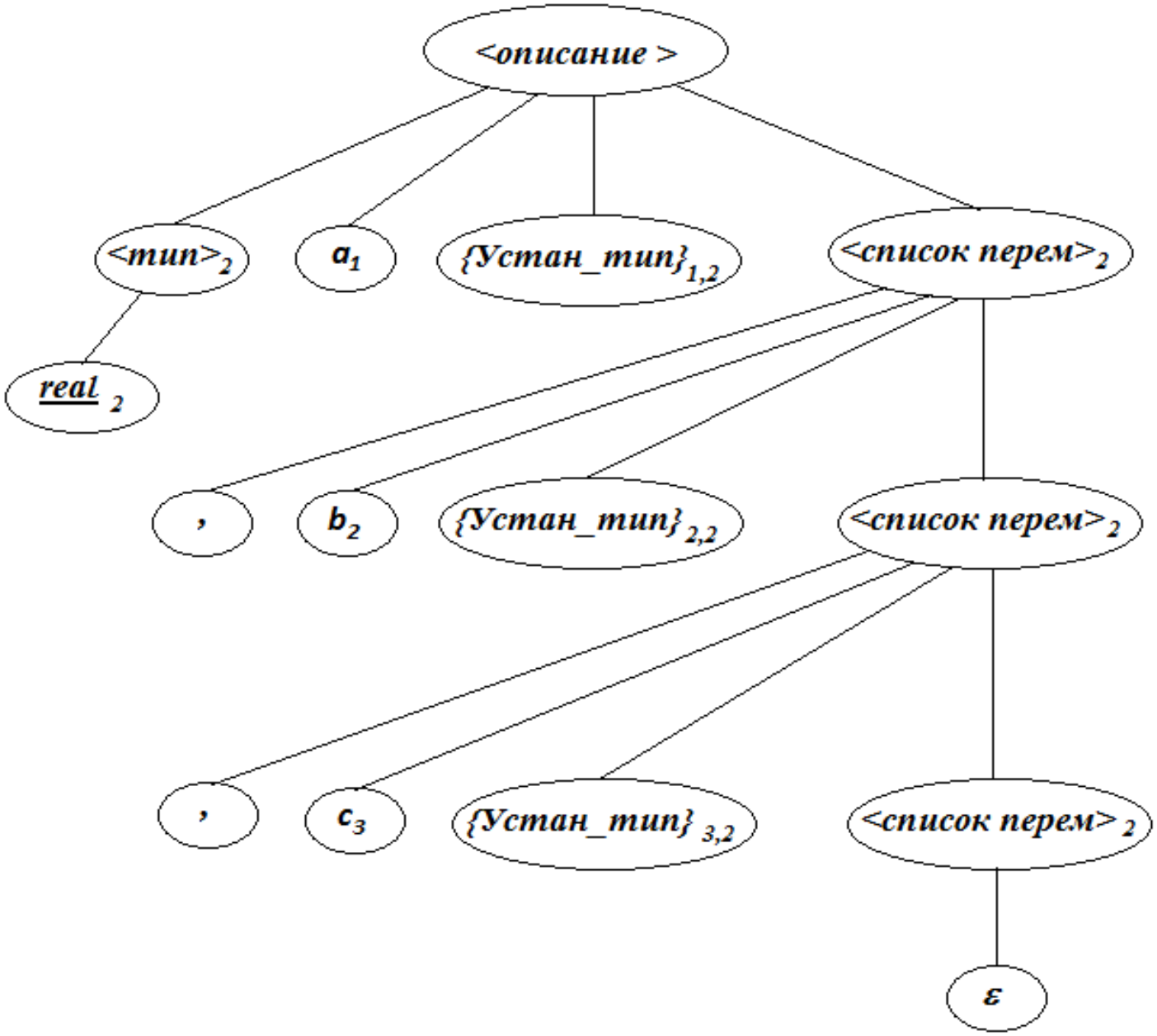
- Нетерминал ***<тип>*** должен иметь атрибут, характеризующий конкретный тип.
- Продукция 1 дополняется правилами вычисления атрибутов и будет выглядеть следующим образом:
- ***<описание> → <тип>$_{t1}$ I$_{p1}$ {Устан_тип $_{p2,t2}$ } <список перем>;***
- ***p2 := p1, t2 := t1***

Атрибутная транслирующая грамматика

- В итоге получим следующую атрибутную грамматику:
-
- ***1. <описание> → <тип>$_{t1}$ I$_{p1}$ {Устан_тип $_{p2,t2}$} <список перем>$_{t3}$;***
- ***p2 := p1, t2 := t1, t3:=t1***
- ***2. <список перем>$_{t1}$ → , I$_{p1}$ {Устан_тип $_{p2,t2}$} <список перем>$_{t3}$***
- ***p2 := p1, t2 := t1, t3:=t1***
- ***3. <список перем>$_{t1}$ → ε***
- ***4. <тип>$_{t1}$ → int$_1$***
- ***t1 := 1***
- ***5. <тип>$_{t1}$ → real$_2$***
- ***t1 := 2***
- ***6. <тип>$_{t1}$ → bool$_3$***
- ***t1 := 3***

- Построим атрибутное дерево вывода для описания:
-
- ***real a,b,c;***

Атрибутное дерево вывода



Синтезируемые и наследуемые атрибуты

-
-
- Таким образом, в общем случае мы имеем наследуемые атрибуты, значения которых вычисляются при движении по дереву сверху вниз, и синтезируемые атрибуты, значения которых вычисляются при движении по дереву снизу вверх.
-
-
-
-