

# Translation Methods

Лабораторные:

1. perl
2. Ручное построение трансляторов
3. Использование автоматических генераторов трансляторов  
e.g. ANTLR (java), Bison + Yacc (c++), Happy (haskell)
4. Написание автоматического генератора транслятора

1 - рсms, 2-4 - защита

$\Sigma, \Sigma^*, L \subset \Sigma^*$  - формальный язык

Базовый класс формальных языков - регулярные (= автоматные). Для порождения - регулярные выражения, для распознавания - конечные автоматы

Контекстно-свободные языки: КС грамматики / МП автоматы

**токены (лексемы)** - единые неделимые элементы языка ( $\in \Sigma$ )

## Лексический анализ

Первый этап любого разбора - *лексический анализ*

Последовательность символов  $\rightarrow$  последовательность токенов ( $\in \Sigma^*$ )

e.g. арифметические выражения

$$\Sigma = \{n, +, \times, (, )\}$$

$$(2 + 2) \times 2 \rightarrow (n + n) \times n$$

$$n : (0|1|\dots|9)(0|1|\dots|9)^*$$

*жадный лексический анализ на базе регулярных выражений*: пропускаем пробельные символы, смотрим первый непробельный, находим максимальный префикс какого-то возможного токена

1. Проверить, что строка выводится в грамматике  $\Gamma$  // алгоритм КЯК  $O(n^3)$
2. Построить дерево разбора
3. Синтаксически управляемая трансляция

$$\begin{aligned} E &\rightarrow T \\ E &\rightarrow E + T \\ T &\rightarrow F \\ T &\rightarrow T \times F \\ F &\rightarrow n \\ F &\rightarrow (E) \end{aligned}$$

**Атрибуто транслирующие грамматики** - КСГ с добавлением двух элементов: атрибуты и транслирующие символы

**транслирующие символы** - фрагменты кода, которые вставляем в грамматику, которые могут взаимодействовать с атрибутами

$$E \rightarrow E + T \{E_0.v = E_1.v + T.v\}$$

$$T \rightarrow T \times F \{T_0.v = T_1.v + F.v\}$$

Нужно быстрее, чем за куб => накладываем ограничения на грамматики

**Однозначность** - если у любого слова не более одного дерева разбора в этой грамматике // Модификация алгоритма Эрли -  $O(n^2)$

**LL, LR** - грамматики, на которые наложены дополнительные ограничения, чтобы разбор работал за линейное время

$\Gamma$ ,  $w$  на вход

Можем строить дерево разбора сверху вниз - **нисходящая трансляция**. Шаг называется *раскрытие нетерминала*

Снизу вверх - **восходящий разбор**. Шаг - *свёртка*

## Метод нисходящих трансляций для LL грамматик

$s$  - стартовый нетерминал,  $w$  - слово, префикс которого разобран ( $x$ ,  $y$  left)

**LL(k)** - **грамматика** - если достаточно посмотреть на первые  $k$  символов  $y$ , чтобы понять, какое правило применить для нетерминала  $A$

Грамматика  $\Gamma$  называется **LL(1)** грамматикой, если  $s \Rightarrow^* xA\xi \Rightarrow x\alpha\xi \Rightarrow^* xc\eta$   
 $s \Rightarrow^* xA\tau \Rightarrow x\beta\sigma \Rightarrow^* xc\xi$   
 $\alpha = \beta$

**def FIRST:**  $(N \cup \Sigma)^* \rightarrow 2^{\Sigma \cup \{\epsilon\}}$

$c \in FIRST(\alpha) \Leftrightarrow \alpha \Rightarrow^* cx$

$\epsilon \in FIRST(\alpha) \Leftrightarrow \alpha \Rightarrow^* \epsilon$

**e.g.**  $S \rightarrow SS$

$S \rightarrow (S)$

$S \rightarrow \epsilon$

$FIRST(S) = \{c, \epsilon\}$

$FIRST('(S)') = \{(\,, \epsilon)\}$

$FIRST(\epsilon) = \{\epsilon\}$

$FIRST(')') = \{')'\}$

## Алгоритм удаления бесполезных символов

1. Удалить непорождающие символы
2. Удалить недостижимые

### Удаление непорождающих символов

1. Множество непорождающих символов  $Gen = \emptyset$

do {

for  $A \rightarrow \alpha$

if  $\alpha \in (\Sigma \cup Gen)^*$ :

Gen  $\cup = A$

} while Gen change

NonGen =  $N \setminus Gen$

```
1 | A - порождающий, но Алгоритм 1 выбрал как порождающий
2 |
3 | $A \rightarrow \alpha \rightarrow^{k-1} x$
```

## Лемма о рекурсивном вычислении FIRST

$\alpha = c\beta$

$FIRST(\alpha) = \{c\}$

$\alpha = A\beta$

$FIRST(\alpha) = (FIRST(A) \setminus \epsilon) \cup (FIRST(\beta) \text{ if } \epsilon \in FIRST(A))$

$FIRST(\epsilon) = \{\epsilon\}$

## Алгоритм

FIRST: map<N, set< $\Sigma \cup \epsilon$ >>

function getFIRST( $\alpha$ )

if  $\alpha = \epsilon$  return  $\{\epsilon\}$

if  $\alpha[i] \in \Sigma$  return  $\{\alpha[i]\}$

//  $\alpha[0] \in N$

return  $(FIRST[\alpha[0]] \setminus \epsilon) \cup (getFIRST(\alpha[1:]), \text{ if } \epsilon \in FIRST[\alpha[0]])$

## Алгоритм построения FIRST

do {

for  $A \rightarrow \alpha$ :

FIRST[A]  $\cup = getFIRST(\alpha)$

} while FIRST changes

**def** FOLLOW:  $N \rightarrow 2^{\Sigma \cup \{\$ \}}$

$c \in FOLLOW(A) \Leftrightarrow S \Rightarrow^* \alpha A c \beta$

$\$ \in FOLLOW(A) \Leftrightarrow S \Rightarrow^* \alpha A$

## Алгоритм FOLLOW

FOLLOW: map<N, set< $\Sigma \cup \$$ >>

FOLLOW(S) =  $\{\$ \}$

do {

for  $A \rightarrow \alpha$

for B in  $\alpha$

let  $\alpha = \xi B \eta$

$\text{FOLLOW}(B) = \text{FIRST}(\eta) \setminus \epsilon$

if  $\epsilon \in \text{FIRST}(\eta)$

$\text{FOLLOW}(B) \cup = \text{FOLLOW}(A)$

} while FOLLOW changes

## Теорема

$\Gamma$  является LL(1)  $\Leftrightarrow \forall A \rightarrow \alpha, A \rightarrow \beta$ :

1.  $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$
2.  $\epsilon \in \text{FIRST}(\alpha) \Rightarrow \text{FIRST}(\beta) \cap \text{FOLLOW}(A) = \emptyset$

Доказательство:

$\Rightarrow$ ) от противного:

] не (1)

1.  $\exists A \rightarrow \alpha, A \rightarrow \beta, c \in \text{FIRST}(\alpha) \cap \text{FIRST}(\beta)$

$$S \Rightarrow^* xA\sigma \Rightarrow x\alpha\sigma \Rightarrow^* xc\xi\sigma$$

$$S \Rightarrow^* xA\sigma \Rightarrow x\beta\sigma \Rightarrow^* xc\eta\sigma$$

2.  $\epsilon \in \text{FIRST}(\alpha) \cap \text{FIRST}(\beta)$

$$S \Rightarrow^* xA\sigma \Rightarrow x\alpha\sigma \Rightarrow^* x\sigma \Rightarrow x\epsilon\tau$$

$$S \Rightarrow^* xA\sigma \Rightarrow x\beta\sigma \Rightarrow^* x\sigma \Rightarrow x\epsilon\tau$$

] не (2)

...

## Рекурсивный спуск

$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$

Node:

s:  $N \cup \Sigma$

ch: array(Node)

token:  $\Sigma \cup \{\$, \}$

next()

$$\text{FIRST}'(A \rightarrow \alpha) = (\text{FIRST}(\alpha) \setminus \epsilon) \cup (\text{FOLLOW}(A) \text{ if } \epsilon \in \text{FIRST}(\alpha))$$

```

1 Node A() {
2     Node res = Node(A)
3     switch (token)
4         FIRST'(A -> a1):
5         // a1 = X1X2...Xl
6         // X1 in N
7         Node x1 = X1()
8         res.addChild(x1)
9         // X1 in N
10        Node x2 = X2()
11        res.addChild(x2)
12        // X3 in Sigma
13        assert x3 = token or Error()
14        res.addChild(token)
15        next()
16
17        ...
18        // Xl ...
19        ...
20
21        return res
22    FIRST'(A -> a2)
23    ...
24
25    default:
26        Error()
27 }

```

ETF (expression, therm, factor)

Grammar:

$$\begin{aligned}
 E &\rightarrow E + T \\
 E &\rightarrow T \\
 T &\rightarrow T \times F \\
 T &\rightarrow F \\
 F &\rightarrow n \\
 F &\rightarrow (E)
 \end{aligned}$$

	FIRST
E	n, (
T	n, (
F	n, (

FIRST (E + T) = {n, (}

FIRST(T) = {n, (}

**def**  $\Gamma$  называется *леворекурсивной*, если в  $\Gamma : A \Rightarrow^+ A\alpha$

**comment**  $\Gamma$  - леворекурсивная  $\Rightarrow \Gamma \notin LL(1)$

$$\begin{aligned}
A &\Rightarrow \beta \Rightarrow^* A\alpha \\
A &\Rightarrow^* B\xi \Rightarrow \gamma\xi \Rightarrow^* A\alpha \\
A &\Rightarrow^* B\xi \Rightarrow \delta\xi \Rightarrow^* x = cy \\
c &\in (FIRST(\delta)) \setminus \epsilon \cup (FIRST(\xi) \text{ if } \epsilon \in FIRST(\delta)) \\
c &\in FIRST(\gamma) \setminus \epsilon \cup (FIRST(\xi) \text{ if } \epsilon \in FIRST(\gamma))
\end{aligned}$$

$A \rightarrow A\alpha$  - непосредственная левая рекурсия

$A \rightarrow \beta$

$\beta\alpha^*$

Устранение левой рекурсии:

$A \rightarrow \beta A'$

$A' \rightarrow \epsilon$

$A' \rightarrow \alpha A'$

$E \rightarrow E + T^\alpha$

$E \rightarrow T^\beta$

## Грамматика с устранённой непосредственной левой рекурсией

$$\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow \epsilon \\
E' &\rightarrow +TE' \\
T &\rightarrow FT' \\
T' &\rightarrow \epsilon \\
T' &\rightarrow \times FT' \\
F &\rightarrow n \\
F &\rightarrow (E)
\end{aligned}$$

	FIRST	FOLLOW
E	( n	\$ )
E'	+ e	\$ )
T	( n	+ \$ )
T'	* e	+ \$ )
F	( n	* + \$ )

```

1 Node E()
2     Node res = Node(E)
3     switch (token)
4     case n, (:
5         // E -> TE'
6         Node t = T()
7         res.addChild(t)
8         Node e' = E'()
9         res.addChild(e')
10        return res
11
12    default:

```

```

13         Error()
14
15     Node E'()
16         Node res = Node(E')
17         switch (token)
18             case $, ):
19                 // E' -> e
20                 return res
21             case +, e:
22                 // E' -> +TE'
23                 assert token == +
24                 res.addChild(Node(t))
25                 next()
26                 Node t = T()
27                 res.addChild(t)
28                 Node e' = E'()
29                 res.addChild(e')
30                 return res
31
32         default:
33             Error()
34
35         // T and T' are similar with above
36
37     Node F()
38         Node res = Node(F)
39         switch (token)
40             case n:
41                 assert token == n
42                 res.addChild(n)
43                 next()
44                 return res
45             case (:
46                 assert token == (
47                 res.addChild(\()
48                 next()
49                 Node e = E()
50                 res.addChild(e)
51                 assert token == )
52                 res.addChild(Node(\)))
53                 next()
54                 return res

```

$$\begin{array}{l}
 A \rightarrow A\alpha \\
 A \rightarrow \beta \\
 \text{---} \\
 A \rightarrow \beta A' \\
 A' \rightarrow \alpha A' \\
 A' \rightarrow \epsilon
 \end{array}$$

$\beta\alpha^*$

A

switch

FIRST'(A  $\rightarrow \beta_1$ )

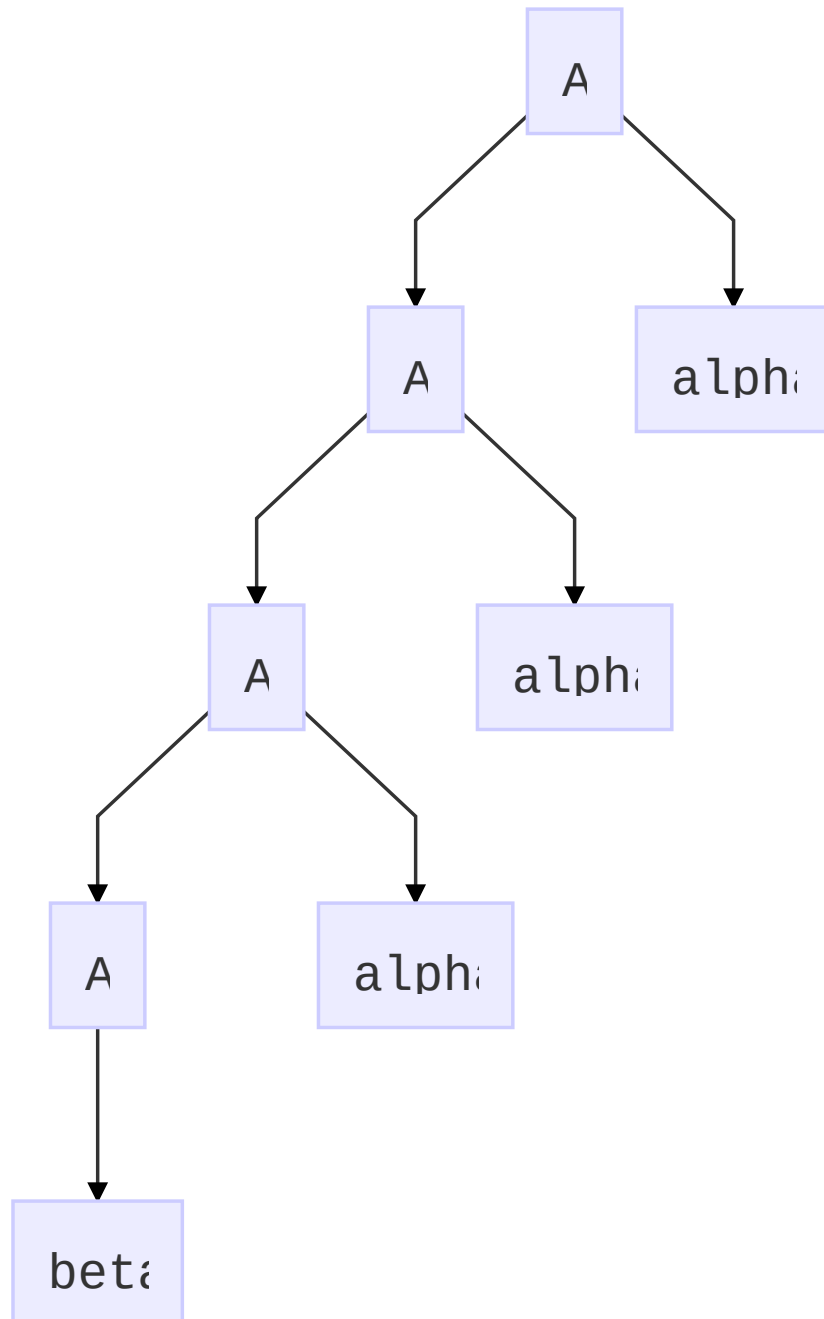
$\beta_1$

$\text{FIRST}'(A \rightarrow \beta_2)$

$\beta_2$

...

while (token  $\in \text{FIRST}'(A \rightarrow A\alpha)$ )





$$A \Rightarrow^+ A\alpha$$

$$A \rightarrow X\alpha, X \in \Sigma \text{ или } \#X > \#A$$

$$A_1, A_2, \dots, A_n, \#A_i = i$$

$$A_1 \rightarrow A_1\alpha$$

$$A_1 \rightarrow \beta$$

$$A_1 \rightarrow \beta A'_1$$

$$A'_1 \rightarrow \alpha A'_1$$

$$A'_1 \rightarrow \epsilon$$

1. Избавиться от  $\epsilon$ -правила

$$A_1 \rightarrow \beta A'_1$$

$$A_1 \rightarrow \beta$$

$$A'_1 \rightarrow \alpha A'_1$$

$$A'_1 \rightarrow \alpha$$

2.  $A_2 \rightarrow A_1\alpha \rightsquigarrow A_2 \rightarrow \xi\alpha$  для всех  $A_1 \rightarrow \xi$  ( $A_2 \rightarrow A_2\beta, A_2 \rightarrow \gamma$ )

$$A_2 \rightarrow A_2\beta$$

$$A_2 \rightarrow \gamma$$

```

1  for i = 1..n
2      for j = 1..i - 1
3          A_i -> A_j alpha
4          for A_j -> xi alpha
5              add A_i -> xi alpha
6          remove A_i -> A_j alpha

```

$$A \rightarrow \alpha\beta$$

$$A \rightarrow \alpha\gamma$$

$$L(\alpha) \neq \{\epsilon\}, \text{ то LL}(1)$$

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta$$

$$A' \rightarrow \gamma$$

## Построение нерекурсивных нисходящих разборов

Стек, управляющая таблица

			$\Sigma$	c		\$
N	A					
					ERROR	
$\Sigma$			SKIP			ERROR
		ERROR				
$\perp$						

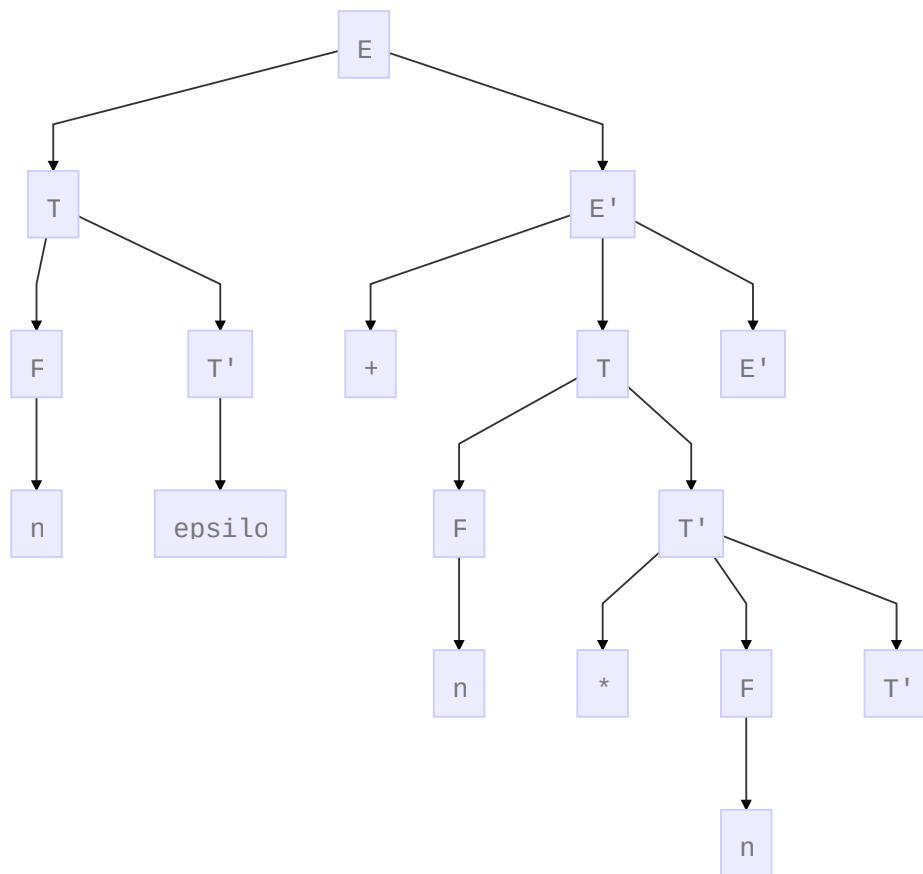
$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow \epsilon \\
 E' &\rightarrow +TE' \\
 T &\rightarrow FT' \\
 T' &\rightarrow \epsilon \\
 T' &\rightarrow \times FT' \\
 F &\rightarrow n \\
 F &\rightarrow (E)
 \end{aligned}$$

	FIRST	FOLLOW
E	( n	\$ )
E'	+ e	\$ )
T	( n	+ \$ )
T'	* e	+ \$ )
F	( n	* + \$ )

	n	+	*	(	)	\$
E	1			1		
E'		2			3	3
T	4			4		
T'		6	5		6	6
F	7			8		

пустые ячейки соответствуют ошибке

e.g. to parse: 2 + 2 \* 2  
 tree:



## Атрибутно-транслирующие грамматики (АТГ)

$N, S \in N; \Sigma; P$  - правила

Расширим определение грамматики

$N$  &  $\Sigma$  определяется в  $Z$

**атрибуты**

$\Sigma, N$

0. имя
  1. тип
  2. значение (может быть не определено)
  3. правило вычисления
- S-атрибуты - только присваивание атрибута

Атрибуты бывают:

### 1. Синтезируемые атрибуты

Если его значение зависит только от поддерева, в том числе, когда этот атрибут - атрибут терминала и его значение на этапе лексического анализа

### 2. Наследуемый атрибут

Значение зависит от родителей или братьев

L-атрибутная

**Транслирующий символ** - специальный нетерминал, у которого единственное правило раскрыть его в  $\epsilon$  и которого есть связанный с ним код, внутри которого мы можем работать с атрибутами

Могут быть **именными** и **анонимными**

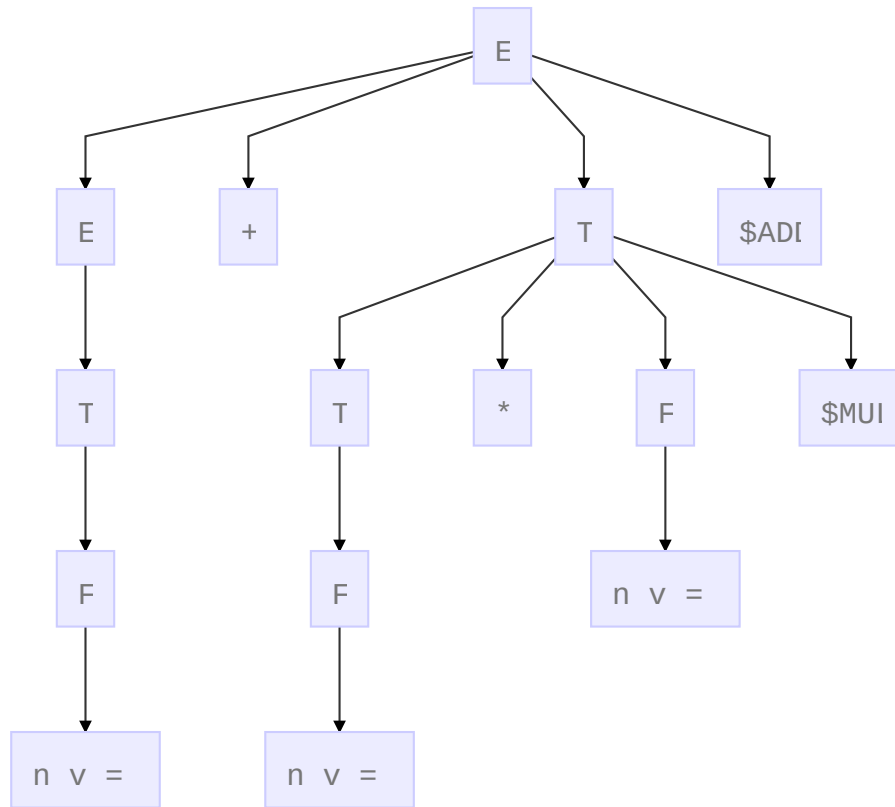
$E \rightarrow E + T$	$\$MUL\ op1 = T_1.v$ $\$MUL\ op2 = F.v$ $T_0.v = \$MUL\ res$
$E \rightarrow T$	$E.v = T.v$
$T_0 \rightarrow T_1 \times_2 F_3$	$\$MUL\ op1 = T_1.v$ $\$MUL\ op2 = F.v$ $T_0.v = \$MUL\ res$
$T \rightarrow F$	$T.v = F.v$
$F \rightarrow n$	$F.v = n.v$
$F \rightarrow (E)$	$F.v = E.v$

1	$\$MUL\ \{$
2	$res = op1 * op2$
3	$\}$

$$\$MUL \left\{ \begin{array}{l} op1 \text{ наследуемый} \\ op2 \text{ наследуемый} \\ res \text{ синтезируемый} \end{array} \right.$$

1	$\$ADD\ \{$
2	$add = op1 + op2$
3	$\}$

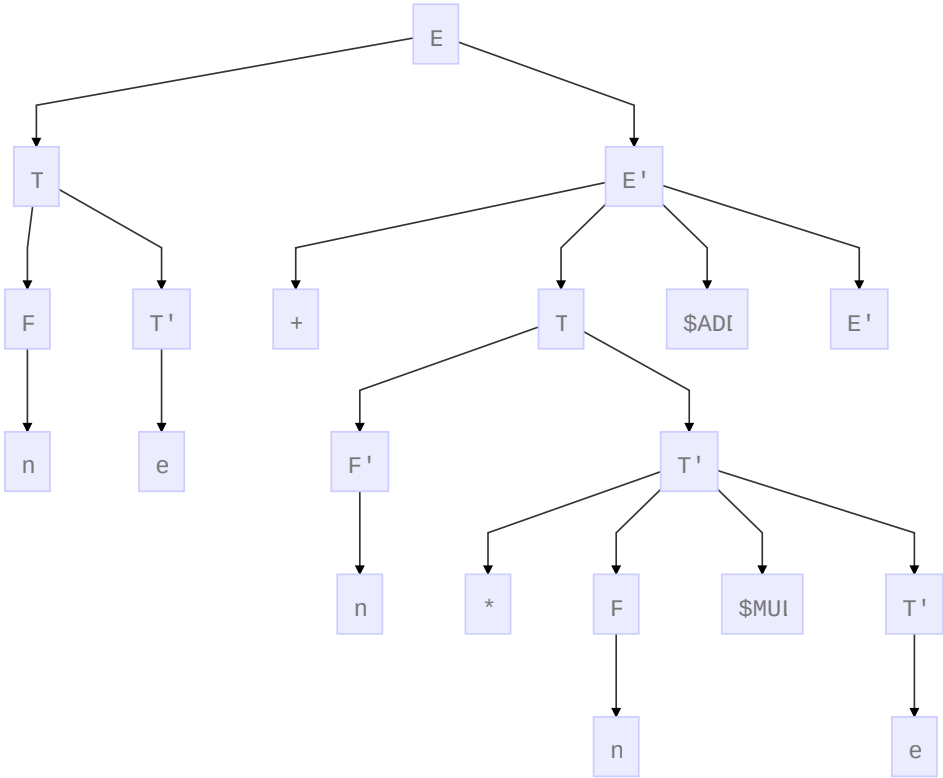
E	v	
T	v	
F		синтезируемый
n		синтезируемый



$E \rightarrow TE'$		$E'.a = T.v$ $E.v = E'.v$
$E' \rightarrow +TE'$	\$ADD E'	$\$ADD\ op1 = E'_0.a$ $\$ADD\ op2 = T.v$ $E'_4.a = \$ADD.res$
$E' \rightarrow \epsilon$		$E'.v = E'.a$
$T \rightarrow FT'$		$T'.a = E'.a$ $T.v = T'.v$
$T' \rightarrow \times FT'$	\$MUL T'	$\$MUL\ op1 = T'_0.a$ $\$MUL\ op2 = F.v$ $T'_4.a = \$MUL\ res$
$T' \rightarrow \epsilon$		$T'.v = T'.a$
$F \rightarrow n$		$F.v = n.v$
$F \rightarrow (E)$		$F.v = E.v$

E	v
T	v
F	v синтезируемый
n	v синтезируемый
E'	a наследуемый v синтезируемый
T'	a наследуемый v синтезируемый

2 + 3 \* 4



```
1 E'(a: int): int
2   switch
3     case // -> e
4       return a
5     case // +T $ADD E'
6       skip +
7       T.v = T()
8       $ADD.res = $ADD(a, T.v)
9       E'.v = E'($ADD.res)
10      return E'.v
```

```

11
12 E(): int
13     switch
14         case
15             T.v = T()
16             E'.v. = E'(T.v)
17             return E'.v
18
19 $ADD(op1, op2: int): int
20     return op1 + op2
21
22 // alternative:
23 Node E'(a)
24     Node res = Node(E, atr = {a.a})
25
26     switch
27         -> e
28             res.v = res.a
29             return res
30         -> +TE'
31             skip +
32             T = T()
33             E'4.a = res.a + T.v
34             E' = E'(E'4.a)
35             res.v = E'.v
36             return res

```

### Регистровые машины и Стековые машины

операции регистровых машин: load загрузить значение и store выгрузить в память  
 преимущество перед регистровыми, в регистровых конечное количество регистров, здесь есть стек и операции push, pop

---

Непосредственная левая рекурсия

$A \rightarrow A\alpha$

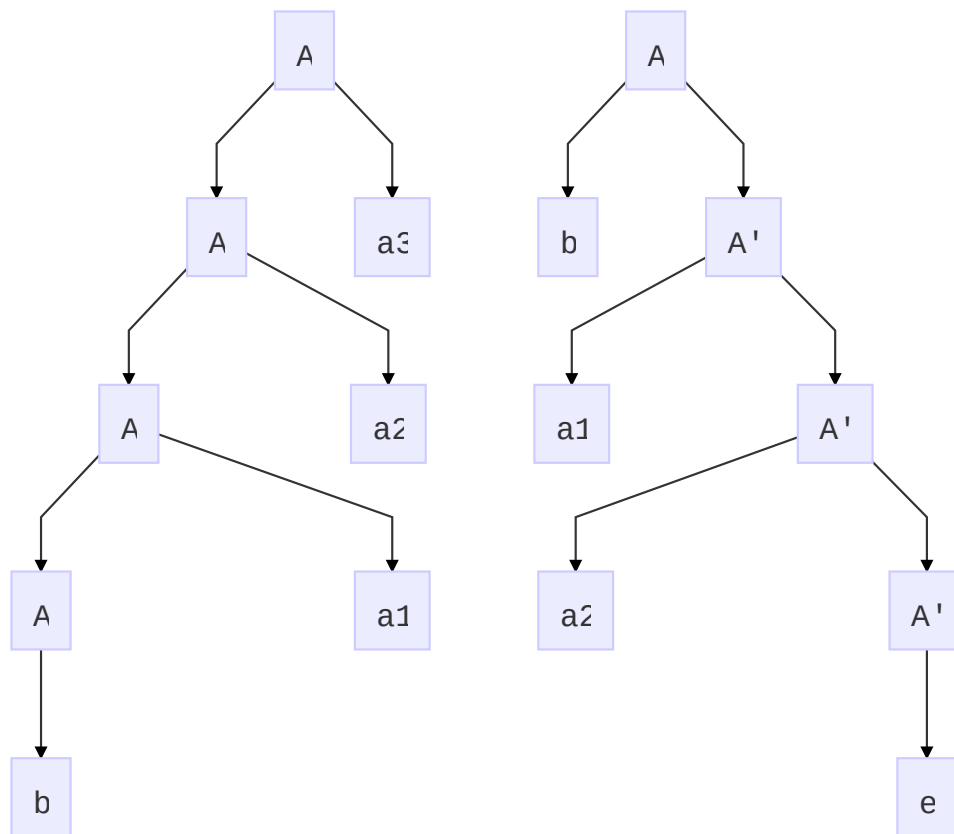
$A \rightarrow \beta$

$x$  - синтезируемый атрибут  $A$

$A \rightarrow \beta A'$

$A' \rightarrow \epsilon$

$A' \rightarrow \alpha A'$



$A' x$  - соответствует  $Ax$  - синтезируемый  
 $a$  - аккумулятор - наследуемый

$A \rightarrow \beta A'$        $A' a = f(\beta)$

$A' \rightarrow \epsilon$

$A' \rightarrow \alpha A'$

$A s$  - синтезируемый атрибут

$a$  - наследуемый атрибут

```

1  A(a) -> s
2      switch ( )
3          ...
4          // A -> a
5          s = f(alpha)
6          // alpha_k = B
7          B(<->)
  
```

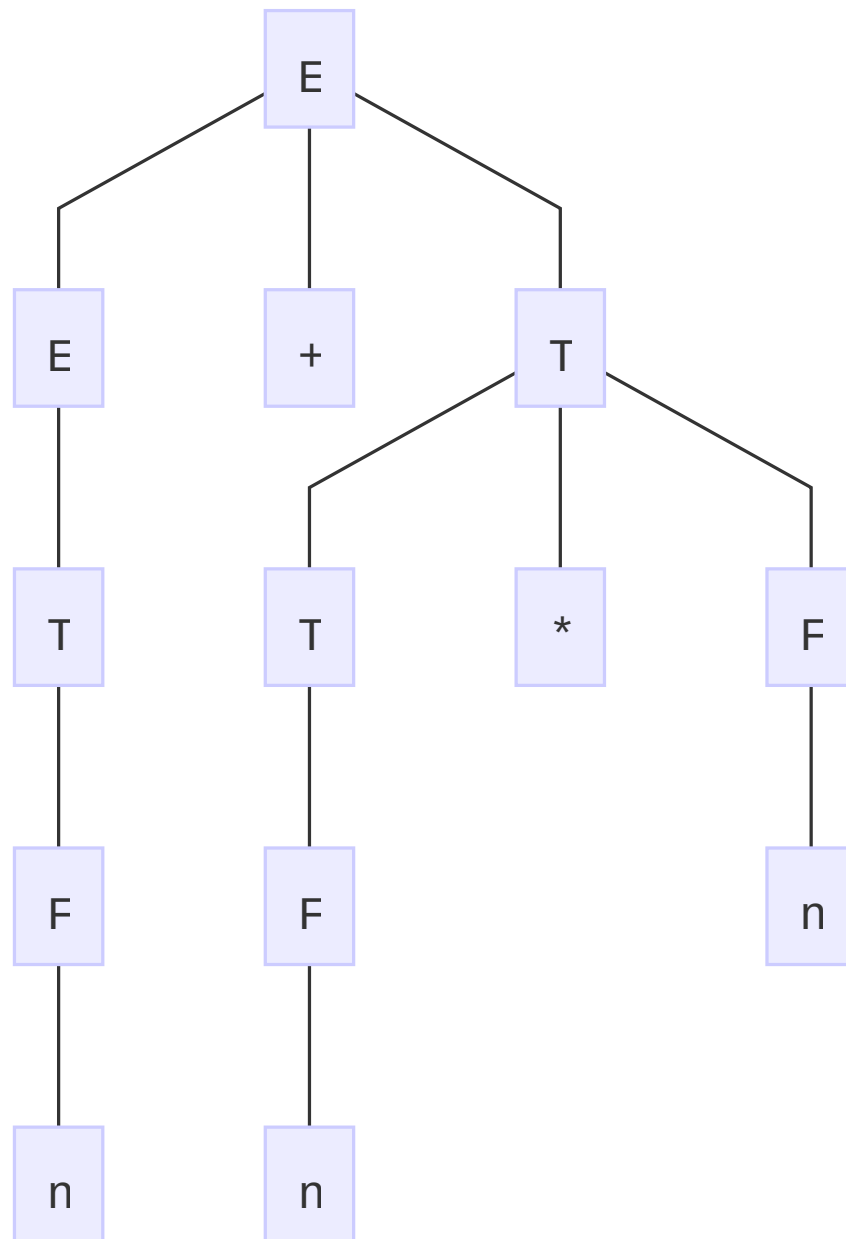
Но вообще генерируются парсеры со стеком

## Восходящий разбор



перенос - свёртка

shift - reduce



### LR - анализ

LR(0) редко используется, есть LR(1)

LR -> SLR (Simple LR) -> LALR -> LR(1)

$$\eta Bu \Rightarrow \eta \beta u = \xi A t \Rightarrow \xi \alpha t = \omega$$

$$\gamma = \xi t \quad S \Rightarrow^* \gamma \quad \xi \in (\Sigma \cup N)^*, t \in \Sigma^*$$

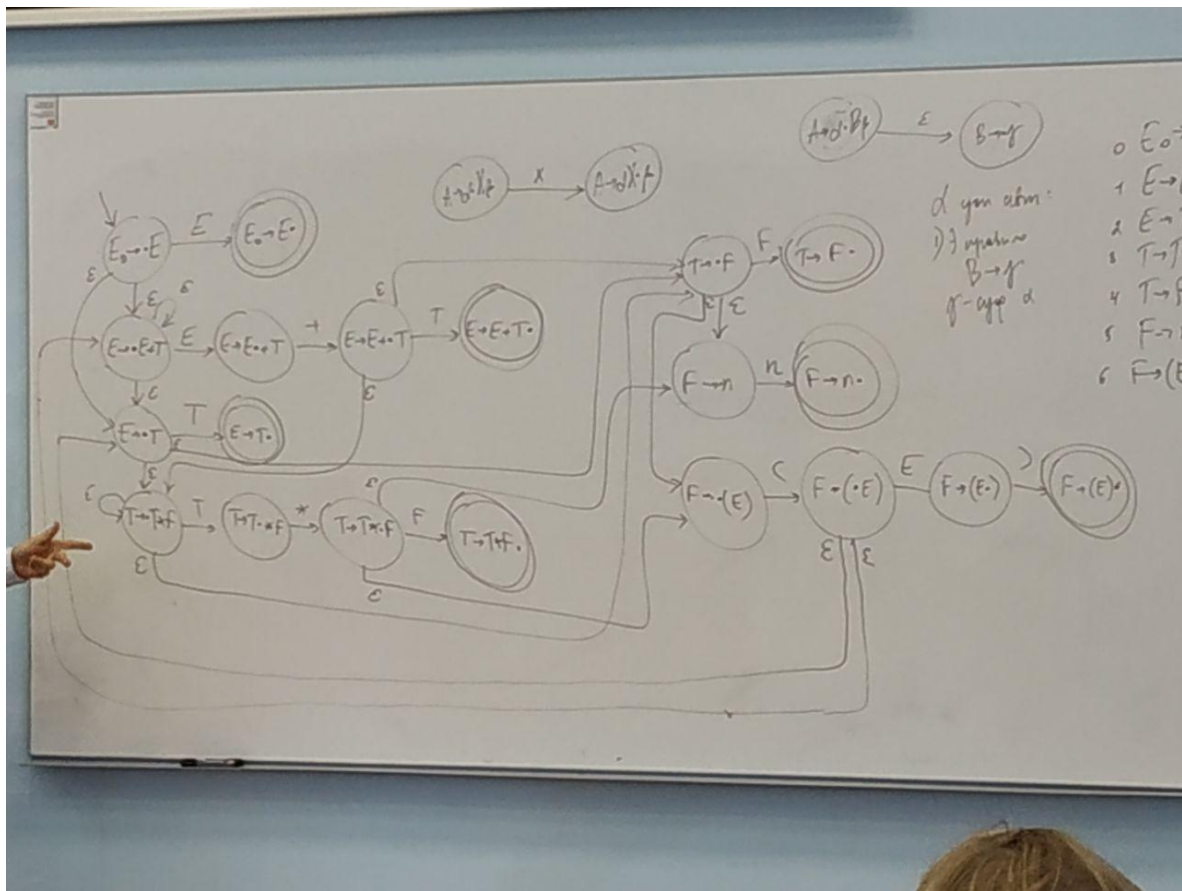
$\alpha$  - подстрока  $\gamma$

$\gamma = \xi' \alpha t'$   $\xi'$  - подстрока  $\xi$ ,  $t'$  - суффикс  $t$

$$S \Rightarrow^* \xi' A t' \Rightarrow \xi' \alpha t' = \xi t$$

Ситуации (items)

LR(1) - ситуация ( $A \rightarrow \alpha, K \in \{0, \dots, |\alpha|\}$ )



## Теорема

Если строка альфа допускается автоматом, построенным по этим правилам, то:

1. Существует правило  $B \rightarrow \gamma$ ,  $\gamma$  - суффикс  $\alpha$

	E	T	F	(	)	n	+	*
1 $[E \rightarrow \cdot E] [E \rightarrow E \cdot T] [E \rightarrow T \cdot] [T \rightarrow \cdot T * F] [T \rightarrow T \cdot F] [T \rightarrow F \cdot] [F \rightarrow \cdot (E)] [F \rightarrow ( \cdot E)]$	2	3	4	5		6		
2 $[E \rightarrow \cdot E] [E \rightarrow E \cdot T]$							7	
3 $[E \rightarrow T \cdot] [T \rightarrow \cdot T * F]$								8
4 $[T \rightarrow F \cdot]$								
5 $[F \rightarrow \cdot (E)] [F \rightarrow (E \cdot)] [F \rightarrow (E) \cdot] [T \rightarrow T \cdot F] [T \rightarrow F \cdot] [F \rightarrow \cdot (E)] [F \rightarrow ( \cdot E)]$	9	3	4	5		6		
6 $[F \rightarrow n \cdot]$								
7 $[E \rightarrow E \cdot T] [T \rightarrow \cdot T * F] [T \rightarrow T \cdot F] [F \rightarrow \cdot (E)] [F \rightarrow ( \cdot E)]$	10	4	5			6		
8 $[T \rightarrow T \cdot F] [F \rightarrow \cdot (E)] [F \rightarrow ( \cdot E)]$			11	5		6		
9 $[F \rightarrow (E \cdot)] [E \rightarrow E \cdot T]$					12		7	
10 $[E \rightarrow E \cdot T] [T \rightarrow T \cdot * F]$								8
11 $[T \rightarrow T * F \cdot]$								
12 $[F \rightarrow (E \cdot)]$								1