Вывод

$$Ex \rightarrow Nat \mid (Ex) \mid Ex + Ex \mid Ex * Ex$$

- **Ex**
- **Ex** + Ex
- Nat + **Ex**
- Nat + **Ex** * Ex
- Nat + Nat * **Ex**
- Nat + Nat * Nat

Левосторонний вывод

- **Ex**
- Ex + **Ex**
- Ex + Ex * **Ex**
- Ex + **Ex** * Nat
- **Ex** + Nat * Nat
- Nat + Nat * Nat

Правосторонний вывод

- **Ex**
- **Ex** * Ex
- **Ex** + Ex * Ex
- Nat + **Ex** * Ex
- Nat + Nat * **Ex**
- Nat + Nat * Nat

Еще один

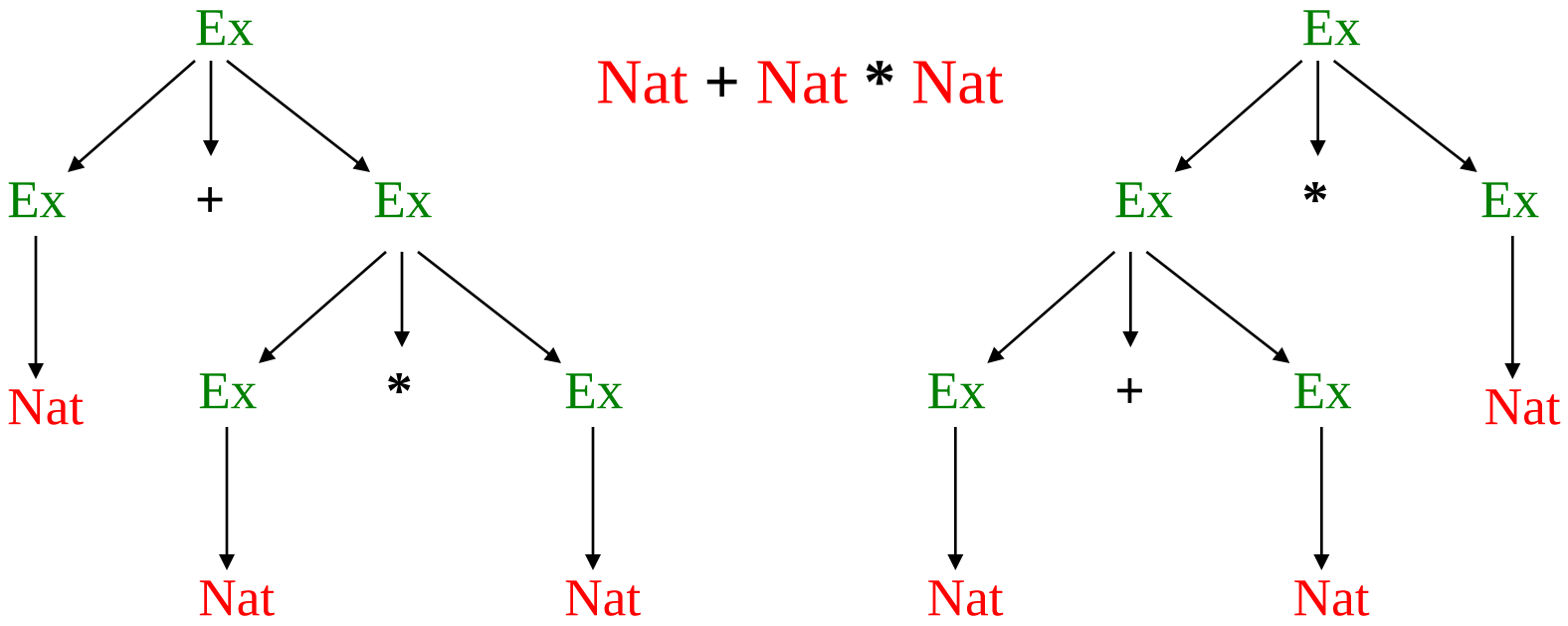
Левосторонний вывод?

Неоднозначные грамматики

- Неоднозначность: если существует два или более различных синтаксических дерева для одной сентенциальной формы (цепочки):

Ex → Nat | **(Ex)** | **Ex** + **Ex** | **Ex** * **Ex**

Nat + Nat * Nat



Nat + (Nat * Nat)

(Nat + Nat) * Nat

Неоднозначные грамматики (2)

- неоднозначность - это **свойство грамматики**, а не языка
- Проблема, порождает ли данная КС-грамматика однозначный язык (т.е. существует ли эквивалентная ей однозначная грамматика), является **алгоритмически неразрешимой**
- некоторые виды правил вывода, которые приводят к неоднозначности:

$A \rightarrow AA \mid \alpha$

$A \rightarrow A\alpha A \mid \beta$

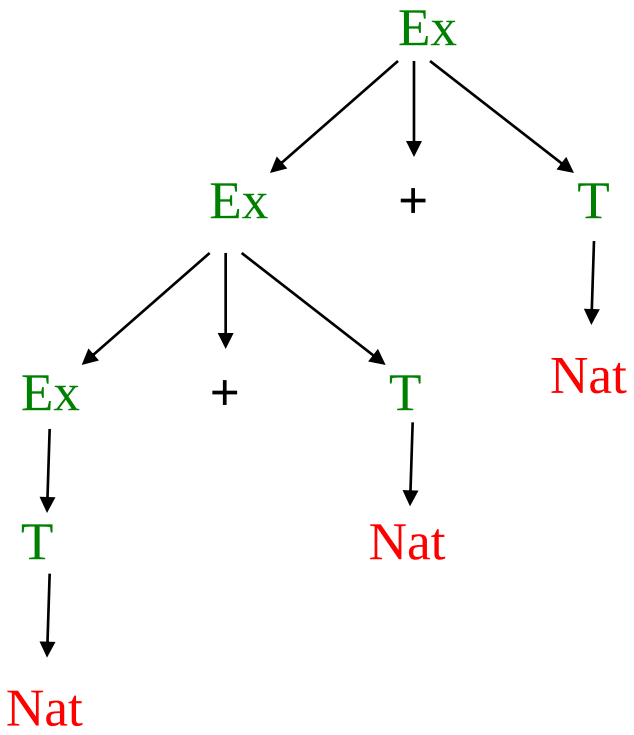
$A \rightarrow \alpha A \mid A\beta \mid \gamma$

$A \rightarrow \alpha A \mid \alpha A\beta A \mid \gamma$

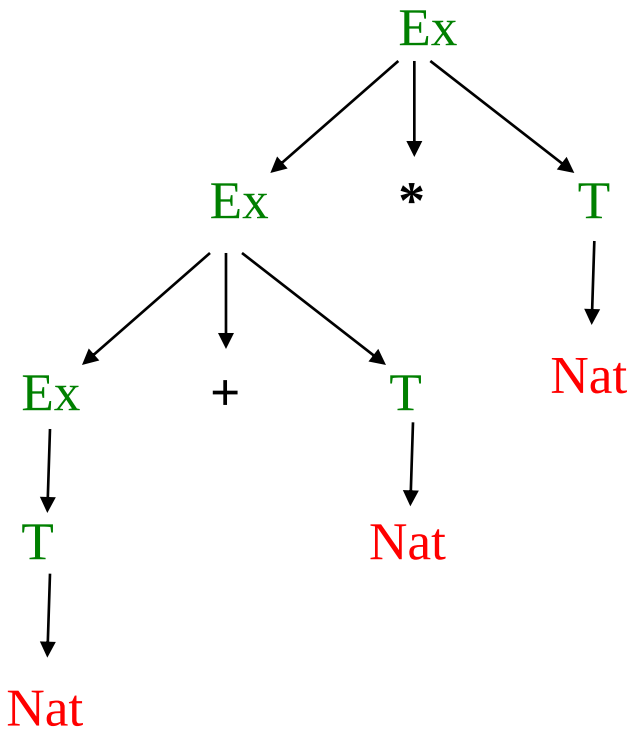
Однозначная грамматика

Ex → **T** | **Ex** + **T** | **Ex** * **T**
T → Nat | **(Ex)**

Nat + Nat + Nat



Nat + Nat * Nat



Контекстно-свободные грамматики

- Символ A
  - *леворекурсивен* *если* $A \to A\ \alpha$
  - *праворекурсивен* *если* $A \to \alpha\ A$
  - *самовыводимый* *если* $A \to \alpha\ A\ \beta$

  где α и β непустые цепочки. (рекурсивные правила)
- Преимущество КС грамматик состоит в самовыводимости (бесконечный язык)

  $S \to (S) \mid ( )$
- Возможно, как альтернатива, сочетание левой и правой рекурсии
  - e.g. $S \to (A$
  - $A \to) \mid S)$

нисходящий и восходящий СА

Ex → **Nat** | (**Ex**) | **Ex** + **Ex** | **Ex** * **Ex**

Нисходящий СА (Top-down):

– “Предсказывающий” анализатор

– Начинаем с начального символа

– Пытаемся вывести цепочку (предложение)

Ex
Ex + **Ex**
Nat + **Ex**
Nat + **Nat**

Восходящий СА (Bottom)-up:

– Начинаем разбор цепочки

– Пытаемся получить начальный символ

– Применяем правила в обратном порядке (правая часть заменяется на левую)

Nat + **Nat**
Ex + **Nat**
Ex + **Ex**
Ex

Предиктивный синтаксический анализ

Ex → **Nat** | (**Ex**) | **Nat** + **Ex** | **Nat** * **Ex**

- Нисходящий, цепочка анализируется слева-на-право
- **Предсказывает** какое правило грамматики нужно применить к нетерминалу
- Строит левосторонний вывод

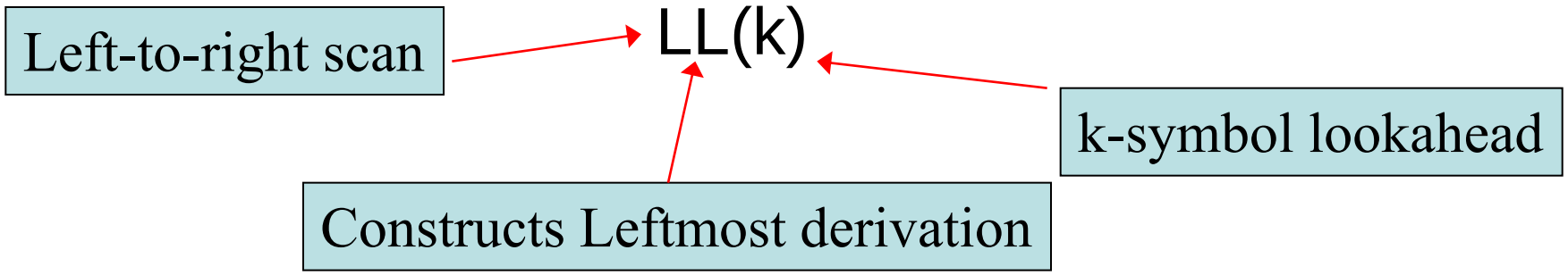
Ex
(**Ex**)
(**Nat** + **Ex**)
(**Nat** + **Nat**)

(	Nat	+	Nat	)

LL(k) Грамматики

Ex → **Nat** | (**Ex**) | **Nat** + **Ex** | **Nat** * **Ex**

- Предсказывающий анализатор работает только с определенным классом грамматик: LL(k)



- Грамматика из примера - LL(2)
- На практике широко используются только LL(1)

Нисходящий СА

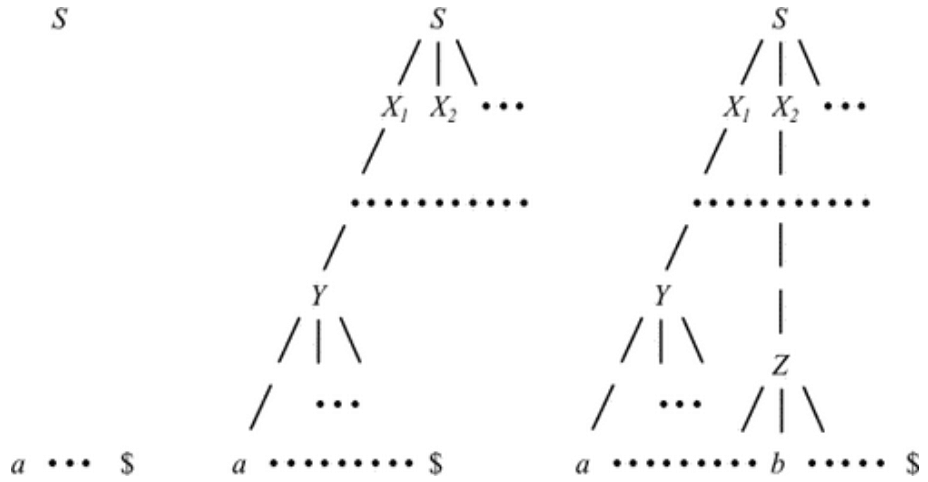
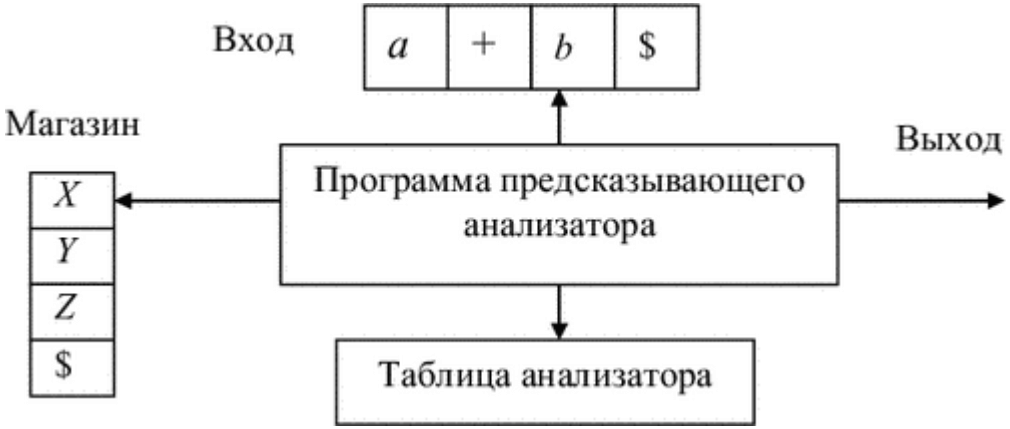


Таблица анализа - двумерный массив **M[A, a]**, где **A** - нетерминал, и **a** - терминал или символ $. Значением **M[A, a]** может быть некоторое правило грамматики или «пусто» (соответствует ошибке)



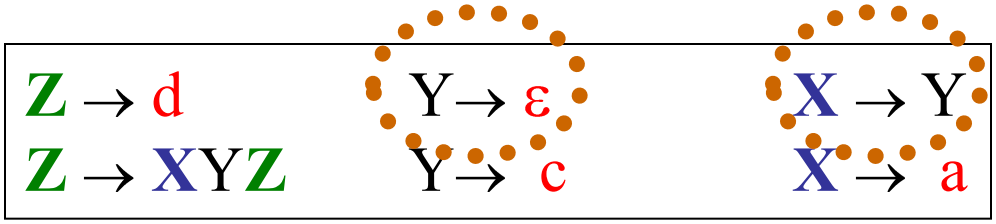
Что нужно для построения таблиц:

- nullable(**X**):
  - «true» если из нетерминала **X** может быть выведена ε
- FIRST(γ):
  - множество терминалов, с которых начинаются строки, выводимые из γ
- FOLLOW(**X**):
  - множество терминалов **t**, которые могут появиться непосредственно справа от **X** в некоторой сентенциальной форме т.е. цепочки вида: $\mathbf{X} \Rightarrow^* \alpha\, \mathbf{X}\, \mathbf{t}\, \beta$

Вычисление nullable(X)

```
for all symbols X do
    nullable(X) := false;
repeat
  change := false;
  for every production X → s1 … sk do
    if nullable(X) = false and
     s1 … sk are all nullable then
          {nullable(X) := true; change := true;}
until change = false;
```

true if k=0 !



Построение FIRST для нетерминалов

```
for all non-terminals X do FIRST(X) := {};
for all terminals t do FIRST(t) := {t};
repeat
 for every production X → s1 … sk do
  { FIRST(X) := FIRST(X) ∪ FIRST(s1);
    for i := 1 to k-1 do
       if nullable(si) then
          FIRST(X) := FIRST(X) ∪ FIRST(si+1);
     else exit;
  }
until no more changes;
```

Z → d	Y → ε	X → Y
Z → XYZ	Y → c	X → a

Y: {c}
X: {a,c}
Z: {a,c,d}

Построение FOLLOW

FIRST:
Y: {c}
X: {a,c}
Z: {a,c,d}

```
for all non-terminals X do FOLLOW(X) := {};
repeat
  for every non-terminal X do
    for each production of the form A → αXβ do
    { FOLLOW(X) := FOLLOW(X) ∪ FIRST(β);
       if nullable(β) then
              FOLLOW(X) := FOLLOW(X) ∪ FOLLOW(A);
    }
until no more changes;
```

[where α and β are possibly empty strings of terminals and non-terminals]

Z → d	Y → ε	X → Y
Z → XYZ	Y → c	X → a

Y: {a,c,d}
X: {a,c,d}
Z: {}

Конструирование LL(1) анализатора

- Построение таблицы анализатора:
  - строки: нетерминалы **X**
  - Столбцы: терминальные символы **c**

  - Правило $\mathbf{X} \rightarrow \gamma$ применимо если
    - $\mathbf{c} \in \text{FIRST}(\gamma)$ *or*
      nullable(γ) and $\mathbf{c} \in \text{FOLLOW}(\mathbf{X})$
- *Если для пары (X, c) применимо более одного правила, то это не LL(1) грамматика.*
- Конфликтов нет – пишем программу.

Нисходящий СА (2)

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | **id**

Нетерминал	Входной символ					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to e$	$E' \to e$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to e$	$T' \to *FT'$		$T' \to e$	$T' \to e$
F	$F \to id$			$F \to (E)$		

Магазин	Вход	Выход
$E\$$	$id + id * id\$$	
$TE'\$$	$id + id * id\$$	$E \to TE'$
$FT'E'\$$	$id + id * id\$$	$T \to FT'$
$id\,T'E'\$$	$id + id * id\$$	$F \to id$
$T'E'\$$	$+id * id\$$	
$E'\$$	$+id * id\$$	$T' \to e$
$+TE'\$$	$+id * id\$$	$E' \to +TE$
$TE'\$$	$id * id\$$	
$FT'E'\$$	$id * id\$$	$T \to FT'$
$id\,T'E'\$$	$id * id\$$	$F \to id$
$T'E'\$$	$*id\$$	
$*FT'E'\$$	$*id\$$	$T' \to *FT'$
$FT'E'\$$	$id\$$	
$id\,T'E'\$$	$id\$$	$F \to id$
$T'E'\$$	$\$$	
$E'\$$	$\$$	$T' \to e$
$\$$	$\$$	$E' \to e$

Реализация анализатора на псевдокоде

```
do
  {X=верхний символ магазина;
   if (X - терминал || X=="$")
      if (X==InSym)
         {удалить X из магазина;
          InSym=очередной символ;
         }
      else error();
   else /*X - нетерминал*/
      if (M[X,InSym]=="X->Y1Y2...Yk")
         {удалить X из магазина;
          поместить Yk,Yk-1,...Y1 в магазин
            (Y1 на верхушку);
          вывести правило X->Y1Y2...Yk;
         }
      else error(); /*вход таблицы M пуст*/
  }
while (X!=$) /*магазин пуст*/
```

Конструирование LL(1) анализатора (2)

- Грамматика является LL(1)-грамматикой тогда и только тогда, когда для каждой пары правил $A \to \alpha$ и $A \to \beta$ выполняются следующие условия:

  $FIRST(\alpha) \cap FIRST(\beta) = \varnothing$

  если $\varepsilon \in FIRST(\alpha)$, то $FIRST(\beta) \cap FOLLOW(A) = \varnothing$

- Если не является, то можно попытаться выполнить

  - устранение левой рекурсии

  $A \to A\alpha_1 | \ldots | A\alpha_n | \beta_1 | \ldots | \beta_m \quad \Rightarrow$

  $A' \to \alpha_1 A' | \ldots | \alpha_n A' | \varepsilon$

  $A \to \beta_1 A' | \ldots | \beta_m A'$

Конструирование LL(1) анализатора (3)

- выполнение левой факторизации

$A \to \alpha\,\beta_1 \mid \alpha\beta_2 \quad \Rightarrow$

$A \to \alpha A'$

$A' \to \beta_1 \mid \beta_2$

Частный случай LL(1) анализатора – метод рекурсивного спуска.

Общие преимущества нисходящих методов:

- Просты для понимания
- Просты для ручной реализации
- Если удалось построить – работают эффективно (без возвратов)

Анализатор типа Сдвиг-Свертка

- Реализуется восходящий анализатор с использованием стека:
- На каждом шаге производится
  - сдвиг: текущий символ помещается в стек, читается следующий символ; или
  - свертка: если верхние m символов стека соответствуют правой части правила, они заменяются на левую часть правила (единственный нетерминал).
- Успех (цепочка принята) если на вершине стека остался единственный символ: начальный символ грамматики

Анализатор типа Сдвиг-Свертка (2)

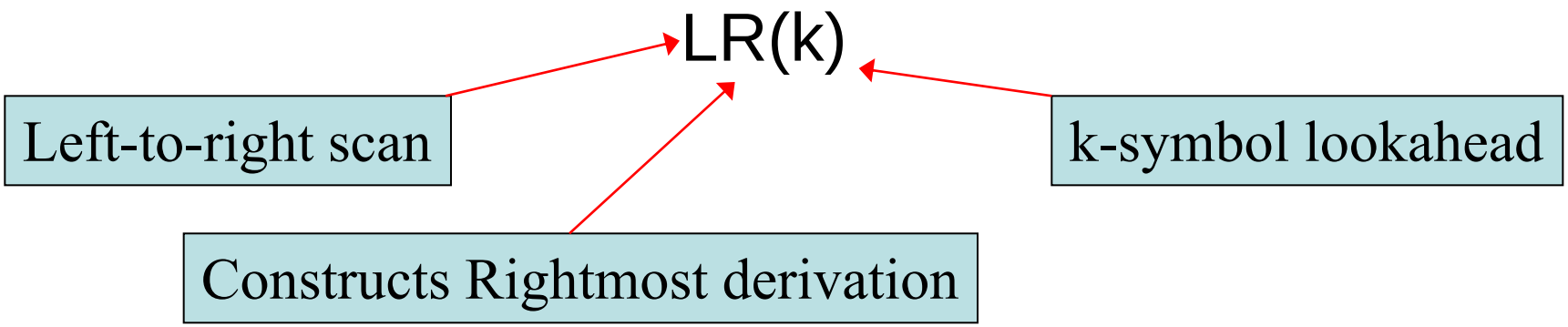
Ex → Nat | (**Ex**) | **Ex** + **Ex** | **Ex** * **Ex**

СТЕК	состояние входа	действие
	Nat + Nat * Nat	←shift
Nat	+ Nat * Nat	↓reduce
Ex	+ Nat * Nat	←shift
Ex +	Nat * Nat	←shift
Ex + Nat	* Nat	↓reduce
Ex + **Ex**	* Nat	←shift
Ex + **Ex** *	Nat	←shift
Ex + **Ex** * Nat		↓reduce
Ex + **Ex** * **Ex**		↓reduce
Ex + **Ex**		↓reduce
Ex		ACCEPT

Алгоритм восходящего разбора

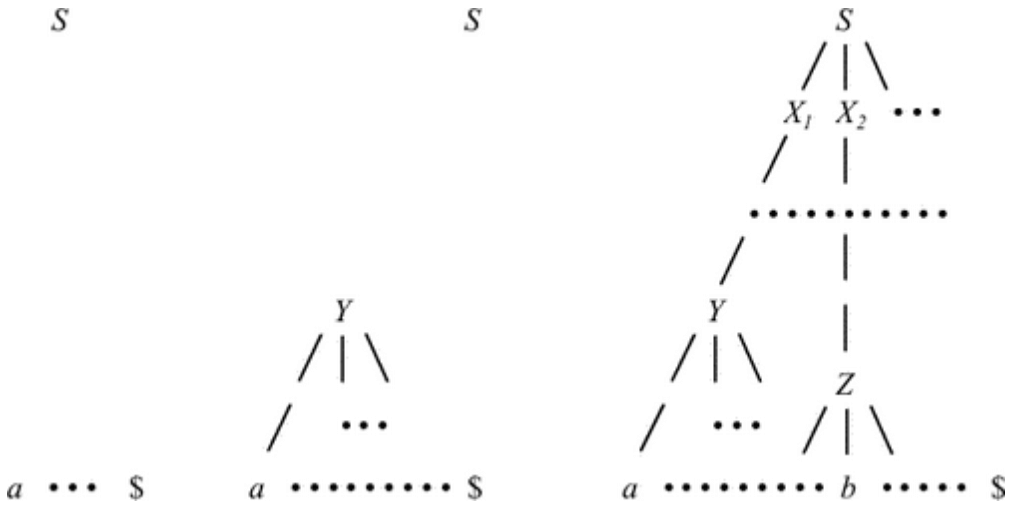
- Необходимо решить на каждом шаге:
  - Сдвиг или свертка?
  - Какое правило использовать при свертке?
  - Что делать при синтаксической ошибке?
- Алгоритмы сложны и трудны для понимания
  - Не применимы для ручной реализации анализатора!
- Мы не будем углубляться в детали восходящих анализаторов (рассмотрим минимум необходимого)

LR(k) Грамматики

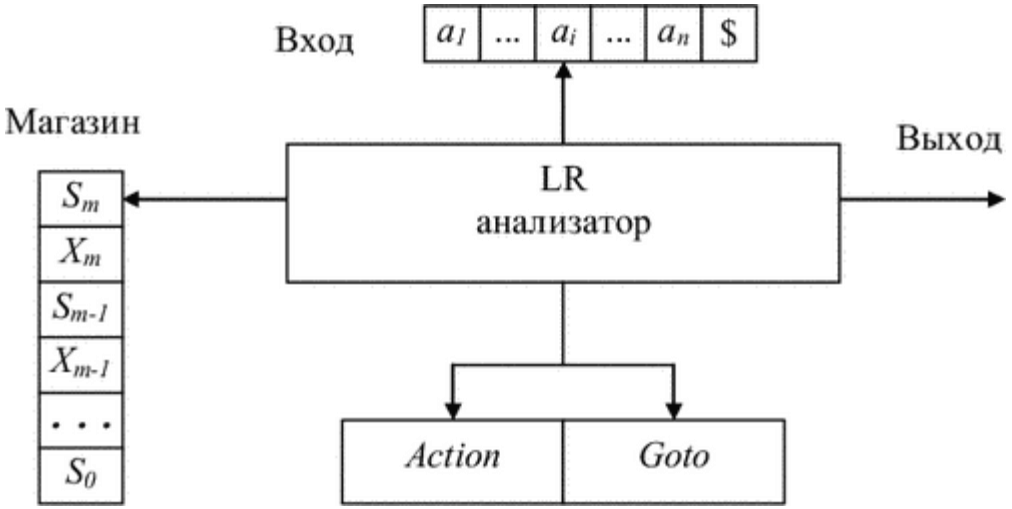


- Используются грамматики типа LR(1) или проще.
- Чаще всего применяются автоматические построения LR анализаторов.

Восходящий СА для LR грамматик



в стеке хранятся строки вида $S_0X_1S_1X_2S_2...X_mS_m$ (S_m - верхушка магазина). Каждый X_i - символ грамматики (терминальный или нетерминальный), а S_i - символ состояния



Восходящий СА для LR грамматик(2)

(1)E → E + T
(2)E → T
(3)T → T * F
(4)T → F
(5)F → **id**

Состояния	*Action*				*Goto*		
	id	+	*	$	E	T	F
0	S6				1	2	3
1		S4		acc			
2		R2	S7	R2			
3		R4	R4	R4			
4	S6					5	3
5		R1	S7	R1			
6		R5	R5	R5			
7	S6						8
8		R3	R3	R3			

Активный префикс	Магазин	Вход	Действие
	0	$id + id * id\$$	сдвиг
id	0 id 6	$+id * id\$$	$F \to id$
F	0 F 3	$+id * id\$$	$T \to F$
T	0 T 2	$+id * id\$$	$E \to T$
E	0 E 1	$+id * id\$$	сдвиг
$E+$	0 E 1 + 4	$id * id\$$	сдвиг
$E+id$	0 E 1 + 4 id 6	$*id\$$	$F \to id$
$E+F$	0 E 1 + 4 F 3	$*id\$$	$T \to F$
$E+T$	0 E 1 + 4 T 5	$id\$$	сдвиг
$E+T*$	0 E 1 + 4 T 5 * 7	$id\$$	сдвиг
$E+T*id$	0 E 1 + 4 T 5 * 7 id 6	$\$$	$F \to id$
$E+T*F$	0 E 1 + 4 T 5 * 7 F 8	$\$$	$T \to T * F$
$E+T$	0 E 1 + 4 T 5	$\$$	$E \to E + T$
E	0 E 1		допуск