

# Контекстно-свободные грамматики

Формализм

Выводы

Форма Бэкуса-Наура

Левые и правые выводы

# Неформальные комментарии

- *Контекстно-свободная грамматика – нотация для описания языков.*
- Она более мощная, чем КДА или РВ, но не может определять все возможные языки.
- Полезна для описания вложенных структур, т.е., скобок в языках программирования.
- Основная идея использования «переменных» – для обозначения множеств строк (т.е., языков).
- Эти переменные определяются рекурсивно, в терминах друг друга.
- Рекурсивные правила (“продукции”) включают только конкатенацию.
- Альтернативные правила для переменных допускают объединение.

# Грамматики

**Опр.** *Порождающая грамматика*  $G$  — это четверка  $\langle T, N, P, S \rangle$ ,

где  $T$  — алфавит терминальных символов (терминалов);

$N$  — алфавит нетерминальных символов (нетерминалов),  $T \cap N = \emptyset$ ;

$P$  — конечное подмножество множества  $(T \cup N)^+ \times (T \cup N)^*$ ;

Пара  $(\alpha, \beta) \in P$  (записывается в виде  $\alpha \rightarrow \beta$ ) называется *правилом вывода*;

$\alpha$  называется *левой частью (головой)* правила,  $\beta$  — *правой частью (телом)* правила.

Левая часть любого правила из  $P$  обязана содержать хотя бы один нетерминал;

$S$  — начальный символ грамматики,  $S \in N$ .

Для записи правил вывода с одинаковыми левыми частями

$$\alpha \rightarrow \beta_1 \quad \alpha \rightarrow \beta_2 \quad \dots \quad \alpha \rightarrow \beta_n$$

используют сокращенную запись  $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ .

# Иерархия языков



Для  $k = 1, 2, 3$  язык типа  $k$  является также и языком типа  $k - 1$  (класс языков типа  $k$  является подклассом класса языков типа  $k - 1$ ).

# Классификация грамматик и языков по Хомскому: Тип 2

**Опр.** Грамматика  $G = \langle T, N, P, S \rangle$  называется **контекстно-свободной** (КС), если каждое правило из  $P$  имеет вид  $A \rightarrow \beta$ , где  $A \in N$ ,  $\beta \in (T \cup N)^*$ .

Заметим, что в КС-грамматиках допускаются правила с пустыми правыми частями. Язык, порождаемый контекстно-свободной грамматикой, называется **контекстно-свободным** языком.

# Формализм КСГ

- *Терминалы* = символы алфавита определяемого языка.
- *Переменные* = *нетерминалы* = конечное множество других символов, каждый из которых представляет язык.
- *Начальный символ* = переменная того языка, который должен быть определен.

# Продукции

- *Продукция* имеет вид:  
переменная (*голова*) -> строка переменных и терминалов (*тело*).
- **Соглашения:**
  - A, B, C,... а также S есть переменные.
  - a, b, c,... - терминалы.
  - ..., X, Y, Z – терминалы или переменные.
  - ..., w, x, y, z –строки только терминалов.
  - $\alpha$ ,  $\beta$ ,  $\gamma$ ,... строки терминалов и/или переменных.

Пример: КС-грамматика для  $\{ 0^n 1^n \mid n \geq 1 \}$

- Продукции:

$S \rightarrow 01$

$S \rightarrow 0S1$

- Базис: 01 есть в языке.

- Индукция: если  $w$  есть в языке, то есть  $0w1$ .



## Пример: Формальное задание КСГ

- Формальное определение КС-грамматики для  $\{0^n 1^n \mid n \geq 1\}$ .
- Терминалы =  $\{0, 1\}$ .
- Переменные =  $\{S\}$ .
- Начальный символ =  $S$ .
- Продукции =
  - $S \rightarrow 01$
  - $S \rightarrow 0S1$

# Примеры грамматик и языков

## Контекстно-свободные

1. Грамматика

$$\begin{aligned} S &\rightarrow aQb / acsb \\ Q &\rightarrow cSc \end{aligned}$$

является контекстно-свободной (неукорачивающей) и порождает КС-язык  $\{(ac)^n (cb)^n \mid n > 0\}$ , который, не является регулярным.

2. Грамматика

$$S \rightarrow aSa / bSb / \varepsilon$$

порождает КС-язык  $\{xx^R, x \in \{a, b\}^*\}$ . Данный язык не является регулярным.

Грамматика не удовлетворяет определению неукорачивающей, но для нее существует эквивалентная неукорачивающая грамматика (см. утверждение 2):

$$\begin{aligned} S &\rightarrow A / \varepsilon \\ A &\rightarrow aAa / bAb / aa / bb \end{aligned}$$

# Выводы – интуитивно

- Мы *выводим* строки в языке КС-грамматики, начиная с начального символа, и многократно заменяя  $A$  телом одной из её продукций.
  - То есть, “продукции для  $A$ ” - это те, которые имеют голову  $A$ .

# Итеративный вывод

- $\Rightarrow^*$  означает “нуль или более шагов вывода.”
- **Базис**:  $\alpha \Rightarrow^* \alpha$  для любой строки  $\alpha$ .
- **Индукция**: Если  $\alpha \Rightarrow^* \beta$  и  $\beta \Rightarrow \gamma$ , то  $\alpha \Rightarrow^* \gamma$ .

## Пример: Итеративный вывод

- $S \rightarrow 01; S \rightarrow 0S1.$
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111.$
- Таким образом,
- $S \Rightarrow^* S; S \Rightarrow^* 0S1; S \Rightarrow^* 00S11; S \Rightarrow^* 000111.$

# Сентенциальные формы

- Любая строка переменных и/или терминалов, которая выводится из начального символа, называется а *сентенциальной формой*.
- Формально,  $\alpha$  - есть сентенциальная форма, т.и.т.т., когда  $S \Rightarrow^* \alpha$ .

# Язык грамматики

**Опр.** *Языком, порождаемым грамматикой*  $G = \langle T, N, P, S \rangle$ , называется множество

$$L(G) = \{ \alpha \in T^* \mid S \Rightarrow \alpha \}.$$

Другими словами,  $L(G)$  — это все цепочки в алфавите  $T$ , которые выводимы из  $S$  с помощью правил  $P$ .

**Опр.** Цепочка  $\alpha \in (T \cup N)^*$ , для которой  $S \Rightarrow \alpha$ , называется *сентенциальной формой* в грамматике  $G = \langle T, N, P, S \rangle$ .

Таким образом, язык, порождаемый грамматикой, можно определить как множество терминальных сентенциальных форм.

# Язык грамматики

- Если  $G$  есть КС-грамматика, то

$L(G)$ , *язык грамматики  $G$* , есть  $\{w \mid S \Rightarrow^* w\}$ .

- **Пример:**  $G$  имеет productions  $S \rightarrow \epsilon$  и  $S \rightarrow 0S1$ .
- $L(G) = \{0^n 1^n \mid n \geq 0\}$ .



# Выводы – Формализм

- Мы говорим, что  $\alpha A \beta \Rightarrow \alpha \gamma \beta$  если  $A \rightarrow \gamma$  - есть продукция.
- **Пример:**  $S \rightarrow 01$ ;  $S \rightarrow 0S1$ .

• 
$$\textcircled{S} \Rightarrow \textcircled{0S1} \Rightarrow \textcircled{00S11} \Rightarrow \textcircled{000111}.$$

# КС-языки

- Язык, определяемый КС-грамматикой называется *контекстно-свободным языком*.
- Существуют КС-языки, которые не являются регулярными, как в предыдущем примере.
- Но не все языки являются КС-языками.
- **Интуитивно**: КС-языки могут считать две вещи, но не три.

# Нотация в форме Бэкуса-Наура

- Грамматики языков программирования часто записывают в БНФ (*форме Бэкуса-Наура*).
- Форма записи грамматик была разработана Бэкусом [18] для описания языка АЛГОЛ в сообщении о языке АЛГОЛ 60
- Наур был редактором сообщения, поэтому форму записи также называют формой Бэкуса - Наура.

# Нотация в форме Бэкуса-Наура

- Отличительные особенности БНФ:
- К зарезервированным символам БНФ относят: '<', '>', '|', ':', '=', '\\';
- Переменные (аналог "нетерминальных символов") пишутся внутри знаков разметки '< ... >';

Например: <предложение>.

- Терминальные символы пишутся "как есть"; Терминалы часто многосимвольные строки, выделяемые жирным шрифтом или подчёркиванием;

Например: **while** или WHILE.

- Альтернативы разделяются знаком '|';
- Левая и правая часть правил разделяются сочетанием "::=";

- Пример
- $\langle \text{число} \rangle ::= \langle \text{чс} \rangle$
- $\langle \text{чс} \rangle ::= \langle \text{чс} \rangle \langle \text{цифра} \rangle \mid \langle \text{цифра} \rangle$
- $\langle \text{цифра} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

## БНФ - нотация – (2)

- Символ  $::=$  часто используется для  $\rightarrow$ .
- Символ  $|$  используется для “или.”
  - Сокращение для списка продукций с одной и той же левой частью.
- **Пример:**  $S \rightarrow 0S1 \mid 01$  сокращение для  $S \rightarrow 0S1$  и  $S \rightarrow 01$ .

# БНФ – нотация – Замыкание Клини

- Символ ... используется для “один или более.”
- **Пример:**  
 $\langle \text{digit} \rangle ::= 0|1|2|3|4|5|6|7|8|9$   $\langle \text{unsigned integer} \rangle ::= \langle \text{digit} \rangle \dots$
- **Трансляция:** Замена  $\alpha \dots$  новой переменной  $A$  и продукциями  
 $A \rightarrow A\alpha \mid \alpha$ .

# Пример: Замыкание Клини

- Грамматика для целых чисел без знака в форме БНФ
- D...
- может быть определена как:

$$U \rightarrow UD \mid D$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$



# БНФ – нотация : Необязательные элементы

- Один или более символов, заключенные в скобки [...] делает их необязательными.
- **Пример:**
- `<statement> ::= if <condition> then <statement> [; else <statement>]`
- **Трансляция:** Заменить  $[\alpha]$  новой переменной  $A$  с productions  $A \rightarrow \alpha \mid \epsilon$ .

## Пример: Необязательные элементы

- Грамматика для for if-then-else может быть определена как:

$S \rightarrow iCtSA$

$A \rightarrow ;eS \mid \epsilon$

$i = \text{if}$

$t = \text{then}$

$e = \text{else}$

$S - \text{statement}$

$C - \text{condition}$

# БНФ – нотация – Группирование

- Используют {...} для выделения группы символов, которые должны использоваться совместно как единое целое.
  - Обычно, за этим следует ... для “один или более.”
- **Пример:** `<statement list> ::= <statement> [{;<statement>}...]`

# Трансляция: Группирование

- Создать новую переменную  $A$  для  $\{\alpha\}$ .
- Одна продукция для  $A$ :  $A \rightarrow \alpha$ .
- Использовать  $A$  вместо  $\{\alpha\}$ .

# Пример: Группирование

$L \rightarrow S \{ ;S \} \dots$

- Заменяем на  $L \rightarrow S [A \dots]$      $A \rightarrow ;S$ 
  - $A$  обозначение для  $\{ ;S \}$ .
- Затем на  $L \rightarrow SB$      $B \rightarrow A \dots \mid \epsilon$      $A \rightarrow ;S$ 
  - $B$  обозначение для  $[A \dots]$  (нуль или более  $A$ ).
- Наконец, заменяем
- $L \rightarrow SB$      $B \rightarrow C \mid \epsilon$      $C \rightarrow AC \mid A$      $A \rightarrow ;S$ 
  - $C$  обозначение для  $A \dots$ .

Расширенная БНФ (РБНФ) Обладает такой же мощностью, что и *БНФ*, но более компактна в записи.

## Фигурные скобки

Выражение с их участием записывается как:

**{<терминал или нетерминал>}<модификатор>**

Фигурные скобки означают, что выражения в них может повторяться от 0 до бесконечности, или согласно модификатору:

- Необязательный модификатор " **\*** " означает, что выражение в скобках может повторяться ноль или бесконечное число раз;
- Модификатор " **+** " означает, что выражение в скобках может повторяться от 1 до бесконечного числа раз;
- Модификатор **(m, n)** означает, что выражение в скобках может повторяться от **m** до **n** числа раз.

# Расширенная БНФ (РБНФ)

## Пример

$\langle U \rangle ::= a\{ab\}$  - цепочка, начинающаяся с  $a$  и содержащая ноль или более (до бесконечности) *цепочек символов*  $ab$  ;

- $\langle U \rangle ::= a\{ab\}^+$  - цепочка, начинающаяся с  $a$  и содержащая от одного до бесконечности повторений цепочки  $ab$  ;

- $\langle U \rangle ::= a\{ab\}(2, 3)$  - содержит цепочки:  $aabab$  и  $aababab$ .

# Расширенная БНФ (РБНФ)

## Квадратные скобки

В них заключено выражение, повторяющееся ноль или один раз.

## Круглые скобки

В правых частях правил оператор конкатенации предшествует оператору выбора. Например,  $AB|C$  означает либо  $AB$  либо  $C$ . Если использовать круглые скобки как метасимвол, мы получим, что  $A(B|C)$  будет означать: либо  $AB$ , либо  $AC$ .

## Диапазон

Чтобы указать, что символы, участвующие в разборе, расположены подряд один за другим, используется символ диапазона "-". Например, запись  $A-Z$  включает в себя все символы, расположенные между  $A$  и  $Z$ , включая эти символы.

**Примечание.** Диапазон нужно использовать только в контексте кодирования символов числами. Поэтому диапазоны всегда привязаны к кодировке. Например: диапазон:  $A-Za-z$  - означает латинские литеры во всех кодировках, диапазон:  $A-Яа-я$  - означает все литеры русского алфавита в кодировках ANSI cp 1251 и Unicode, а  $A-Яа-пр-я$  - все русские литеры в кодировке OEM 866.

## Метасимволы и терминальные символы

Для того чтобы метасимволы:  $(':', '=', '|', '<', '>', '{', '}', '[', ']', '-', '+', '(', ')', '\\$  - могли использоваться как терминальные символы, перед ним вставляется знак  $\\$ .



# Пример

$\langle \text{число} \rangle ::= \{ \langle \text{цифра} \rangle \}^+ \quad \langle \text{цифра} \rangle ::= 0-9$

$\langle \text{врж} \rangle ::= [ \langle \text{врж} \rangle (\backslash + | \backslash -) ] \langle \text{терм} \rangle$

$\langle \text{терм} \rangle ::= [ \langle \text{терм} \rangle (* | /) ] \langle \text{множ} \rangle$

$\langle \text{множ} \rangle ::= [ \langle \text{множ} \rangle ^ ] \langle \text{степ} \rangle$

$\langle \text{степ} \rangle ::= \backslash ( \langle \text{врж} \rangle \backslash ) | \langle \text{идентификатор} \rangle | \langle \text{число} \rangle$

# Деревья синтаксического разбора

Определения

Связь левых и правых выводов

Неоднозначность в грамматиках

# Левые и правые выводы

- Выводы позволяют нам заменять какое-то число переменных в строке.
  - Это ведет к многим различным выводам одной и той же строки.
- Принудительное использование крайней левой переменной (или альтернативно, крайней правой переменной) для замены, позволяет нам избежать этих “отличий без различий.”

# Левые выводы

- Будем говорить, что  $wA\alpha \Rightarrow_{lm} w\beta\alpha$  если  $w$  – строка только терминалов и  $A \rightarrow \beta$  - продукция.
- Также,  $\alpha \Rightarrow_{lm}^* \beta$  если  $\alpha$  становится  $\beta$  в результате последовательности 0 или более  $\Rightarrow_{lm}$  шагов.
- $lm = \text{leftmost}$

## Пример: Левые выводы

- Грамматика сбалансированных скобок:
- $S \rightarrow SS \mid (S) \mid ()$
- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- Таким образом,  $S \Rightarrow_{lm}^* (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$  есть вывод, но не левый вывод.

# Правые выводы

- Будем говорить, что  $\alpha A w \Rightarrow_{rm} \alpha \beta w$  если  $w$  – строка только терминалов и  $A \rightarrow \beta$  - продукция.
- Также,  $\alpha \Rightarrow_{rm}^* \beta$  если  $\alpha$  становится  $\beta$  в результате последовательности 0 или более  $\Rightarrow_{rm}$  шагов.
- $rm = \text{rightmost}$

## Пример: Правые выводы

- Грамматика сбалансированных скобок :
- $S \rightarrow SS \mid (S) \mid ()$
- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Таким образом,  $S \Rightarrow_{rm}^* (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$  не левый и не правый вывод.

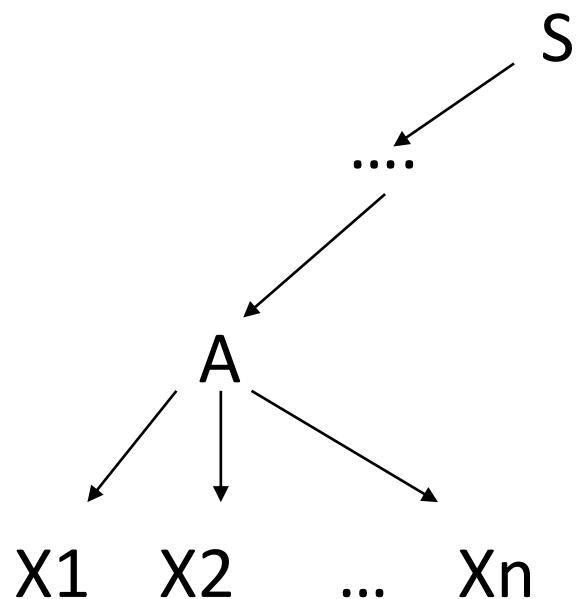
# Деревья синтаксического разбора

- *Деревья синтаксического разбора* – это деревья, помеченные символами соответствующей КС-грамматики.
- **Листья**: помечены терминалами или  $\epsilon$ .
- **Внутренние узлы**: помечены переменными.
  - Дочерние помечены телом продукции для родителя.
- **Корень**: должен быть помечен начальным символом.



# Деревья выводов

- $G = (N, \Sigma, P, S)$
- $A \rightarrow X_1, X_2, \dots, X_n$
- .....
- 



# Дерево вывода

- **Опр.** Помеченное упорядоченное дерево  $D$  называется **деревом вывода** (или **деревом разбора**) в КС-грамматике  $G = (N, \Sigma, P, S)$ , если выполнены следующие условия:
  1. Корень дерева  $D$  помечен  $S$ .
  2. Каждый узел имеет метку из  $N \cup \Sigma \cup \{\varepsilon\}$
  3. Если узел имеет хотя бы одного потомка, метка этого узла – нетерминальный символ
  4. Если  $D_1, D_2, \dots, D_k$  - поддеревья с корнями  $X_1, X_2, \dots, X_k$ , которые являются прямыми потомками, то в множестве правил  $P$  присутствует правило  $X_1 \dots X_k$ . При этом поддерево  $D_i$  может быть узлом с меткой корня  $X_i = \varepsilon$  только в случае, если  $X_i$  - единственный потомок узла  $A$  и в  $P$  присутствует правило  $A \rightarrow \varepsilon$ .

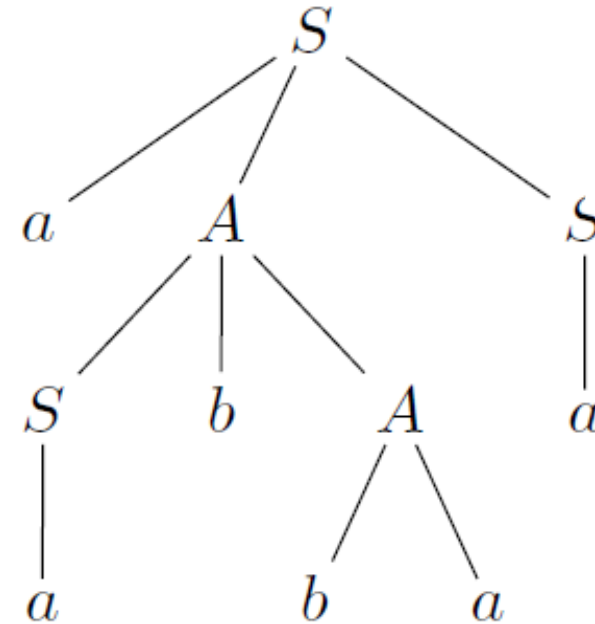
# Пример дерева разбора

Рассм.  $G = (\{S, A\}, \{a, b\}, P, S)$ ,

где

$P: S \rightarrow aAS \mid a$

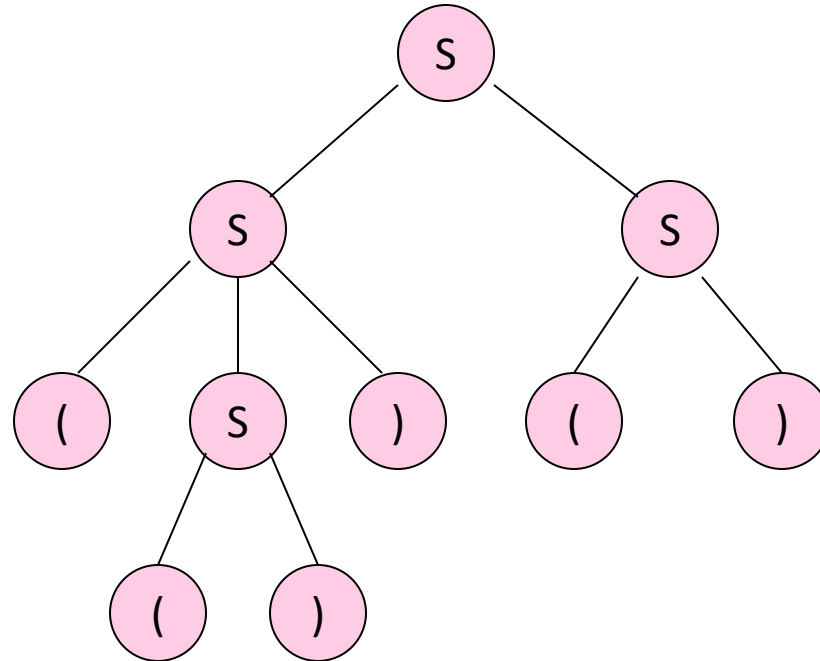
$A \rightarrow SbA \mid ba \mid SS$



$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aabAa \Rightarrow aabbaa$

# Пример: Дерево разбора

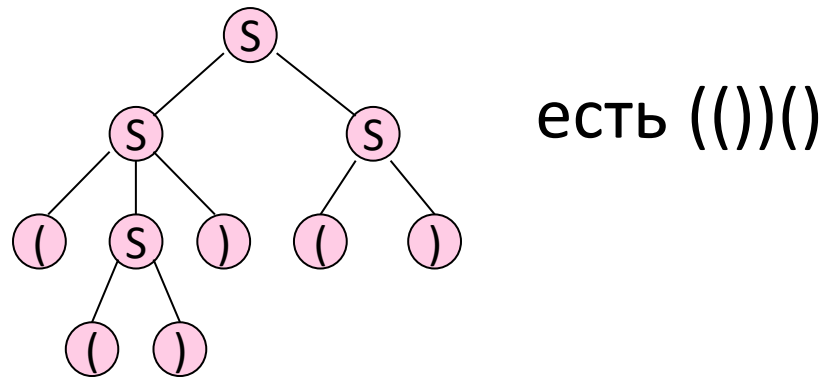
$S \rightarrow SS \mid (S) \mid ()$



# Результат дерева разбора

- **Опр.** Строка, являющаяся конкатенацией меток листьев в порядке слева направо, называется *кроной* дерева вывода (синтаксического разбора).
  - То есть, в порядке определенного обхода.

- **Пример:** Крона



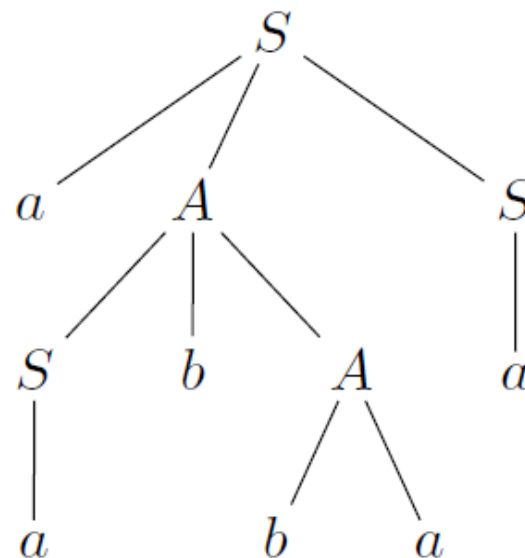
# Сечение, крона сечения

- **Опр.** *Сечением* дерева  $D$  называется такое множество  $C$  вершин дерева  $D$ , что
  - (1) Никакие две вершины из  $C$  не лежат на одном пути в  $D$ ,
  - (2) Каждая вершина из  $C$  принадлежит хотя бы одному пути к корню дерева  $D$ .
  - (3) Ни одну вершину дерева  $D$  нельзя добавить в  $C$ , не нарушив свойства (1).

*Кроной сечения* называют строку – конкатенацию вершин сечения

- **Примеры:**

- Корень дерева есть сечение.
- Множество листьев есть сечение.
- Крона дерева есть сечение.
- Для дерева на рис.:  $(a, A, a)$  -сечение



# Обобщение деревьев разбора

- Далее мы иногда мы будем говорить о деревьях, которые не являются в точности деревьями разбора, т.к. их корень помечен некоторой переменной  $A$ , не являющейся начальным символом.
- Во всём остальном – они выполняют требования к деревьям разбора (листья помечены терминалами или  $\varepsilon$ , внутренний узел и его сыновья образуют правило вывода грамматики).
- Будем называть их *деревьями разбора с корнем  $A$* .

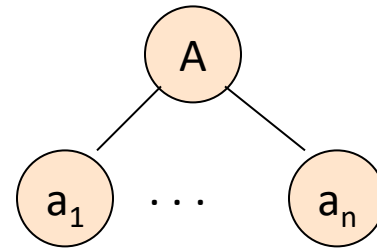
# Деревья разбора, левый и правый выводы

- **Теорема.** Деревья разбора, левый и правый выводы находятся во взаимном соответствии.
- Мы докажем:
  1. Если есть дерево разбора с корнем  $A$  и кроной  $w$ , то  $A \Rightarrow_{lm}^* w$ .
  2. Если  $A \Rightarrow_{lm}^* w$ , то существует дерево разбора с корнем  $A$  и кроной  $w$ .



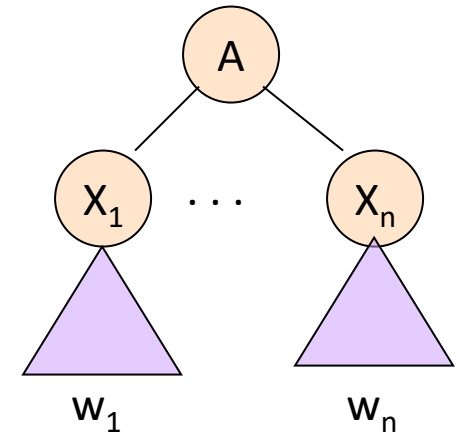
# Доказательство – Часть 1

- Индукция по **высоте** (длине наиболее длинного пути от корня) дерева.
- **Базис**: высота 1. Дерево выглядит так:
- $A \rightarrow a_1 \dots a_n$  должно быть продукцией.
- Таким образом,  $A \Rightarrow_{lm}^* a_1 \dots a_n$ .



# Часть 1 – Индукция

- Предположим (1) выполняется для деревьев высоты  $< h$ , и пусть дерево имеет высоту  $h$ :
- По IH,  $X_i \Rightarrow^*_{lm} w_i$ .
  - Заметим: Если  $X_i$  - терминал, то  $X_i = w_i$ .
- Таким образом,  $A \Rightarrow_{lm} X_1 \dots X_n \Rightarrow^*_{lm} w_1 X_2 \dots X_n \Rightarrow^*_{lm} w_1 w_2 X_3 \dots X_n \Rightarrow^*_{lm} \dots \Rightarrow^*_{lm} w_1 \dots w_n$ .

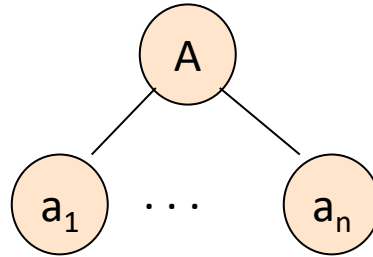


## Доказательство: Часть 2

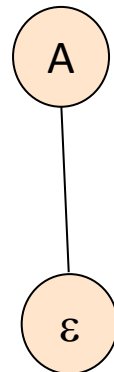
- По данному левому выводу терминальной строки нам нужно доказать существование дерева разбора.
- Доказательство проводится индукцией по длине вывода.

## Часть 2 – Базис

- Если  $A \Rightarrow_{lm} a_1 \dots a_n$  вывод за один шаг, то должно существовать дерево разбора:



- Если  $A \Rightarrow_{lm} \varepsilon$ , то

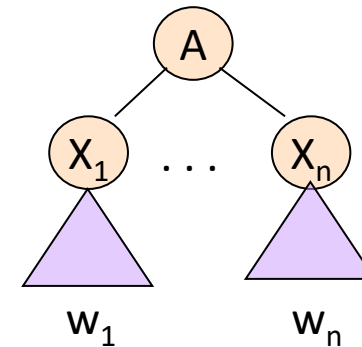


## Часть 2 – Индукция

- **ИН:** Предположим, что (2) выполняется для выводов за число шагов меньшее, чем  $k$ ,
- Пусть  $A \Rightarrow_{lm}^* w$  будет выводом за  $k$  шагов.
- Первый шаг есть  $A \Rightarrow_{lm} X_1 \dots X_n$ .
- **Ключевой момент:**  $w$  может быть разделено так, что первая часть выводится из  $X_1$ , следующая – из  $X_2$ , и.т.д.
  - Если  $X_i$  - терминал, то  $w_i = X_i$ .

## Индукция – (2)

- То есть,  $X_i \Rightarrow^*_{lm} w_i$  для всех  $i$  таких, что  $X_i$  есть переменная и вывод имеет меньше, чем  $k$  шагов.
- По **ИН**, Если  $X_i$  - переменная, то существует дерево разбора с корнем  $X_i$  и кроной  $w_i$ .
- Таким образом, существует дерево разбора:



- Корень даёт первую продукцию вывода,  $X_i$  каждый  $X_i$  есть либо терминал, либо корень дерева с выводом  $w_i$
- Этим доказывается индуктивный шаг и, следовательно, если есть левый вывод  $w$  из  $A$ , то существует дерево разбора с корнем  $A$  и кроной  $w$ .

# Деревья разбора и правые выводы

- Приведенные утверждения имеют зеркальное отражение для правых выводов.
- Если есть правый вывод  $w$  из  $A$ , то существует дерево разбора с корнем  $A$  и кроной  $w$ , и наоборот.
- Доказательство – аналогично.

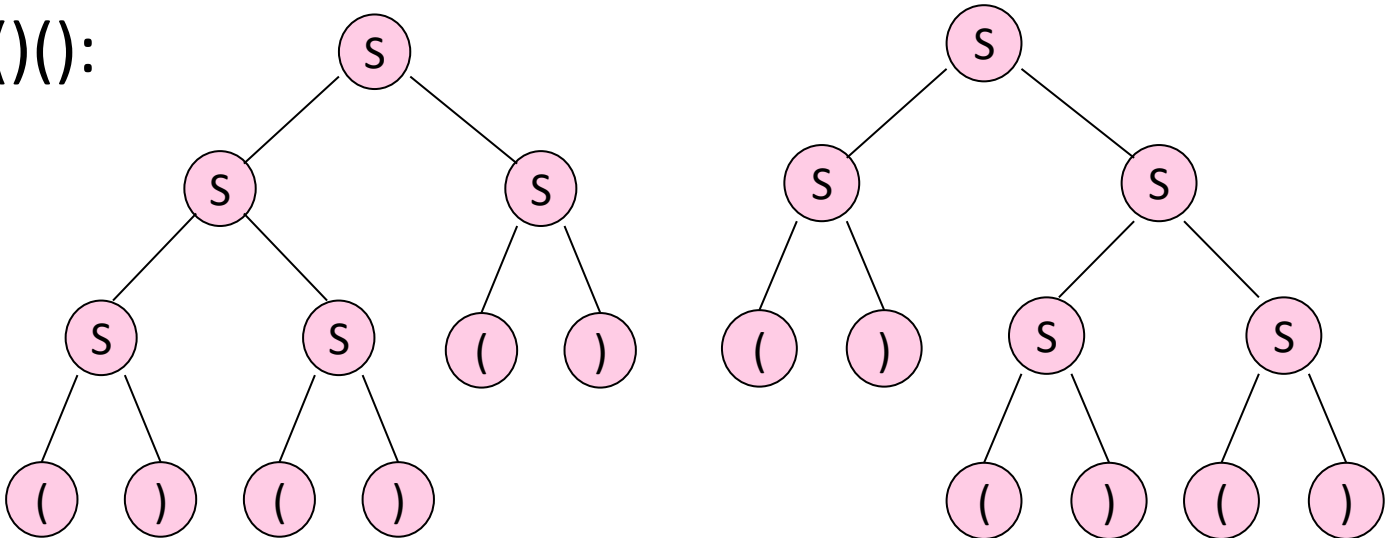
# Деревья разбора и любые выводы

- Доказательство того, что мы можем получить дерево разбора из левого вывода, на самом деле, не зависит от его «левого» типа.
- Первым шагом должен остаться вывод  $A \Rightarrow X_1 \dots X_n$ .
- И  $w$  все ещё можно разделить так, что первая часть выводится из  $X_1$ , следующая выводится из  $X_2$ , и.т.д.
- Таким образом, мы левые и правые варианты вывода могут быть смешаны.



# Неоднозначные грамматики

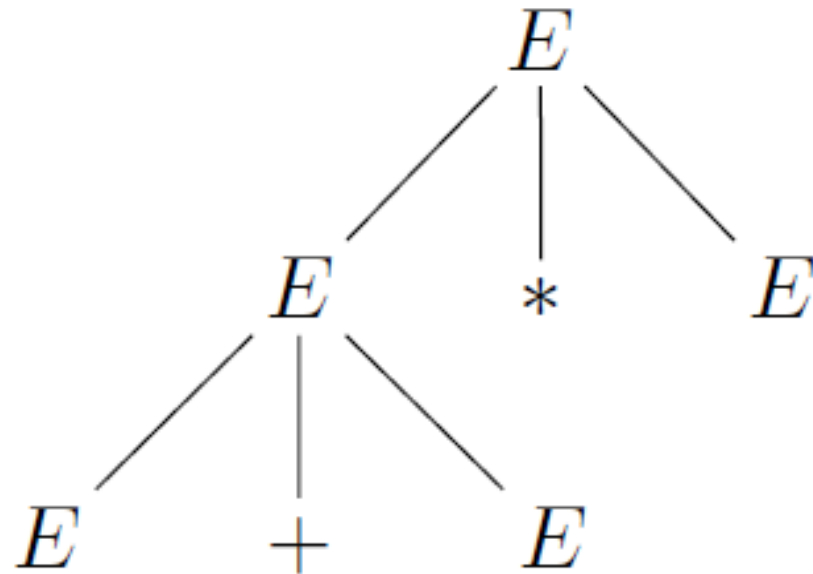
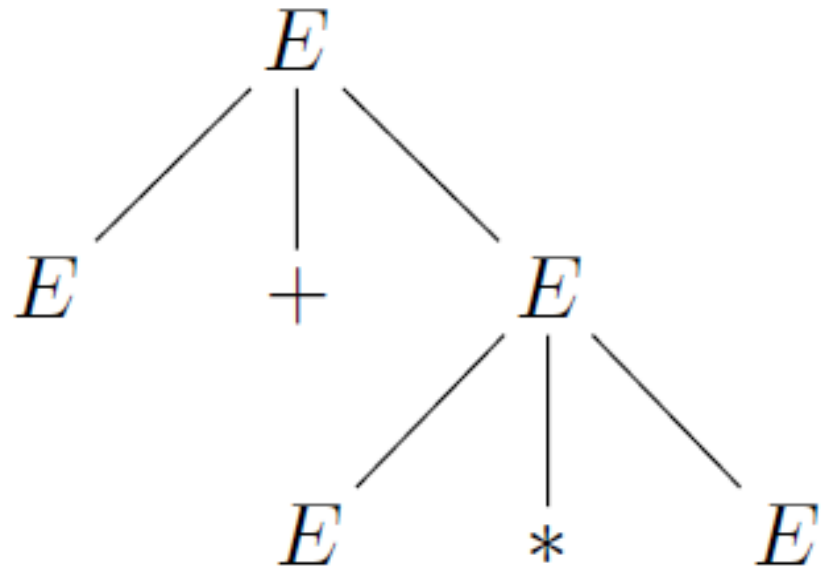
- **Опр.** КС-грамматика является *неоднозначной*, если существует строка языка, которая имеет два или более разных деревьев разбора.
- **Пример:**  $S \rightarrow SS \mid (S) \mid ()$
- Два дерева разбора  $()()()$ :



$$G = (\{E, I\}, \{+, *, (, ), a, b, 0, 1\}, P, E),$$

$$\begin{aligned} E &\rightarrow I \mid E + E \mid E * E \mid (E), \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1. \end{aligned}$$

$$\begin{aligned} E &\Rightarrow E + E \Rightarrow E + E * E, \\ E &\Rightarrow E * E \Rightarrow E + E * E. \end{aligned}$$



- Само по себе существование различных выводов одной и той же цепочки ещё не означает неоднозначности грамматик.

$$G = (\{E, I\}, \{+, *, (, ), a, b, 0, 1\}, P, E),$$

$$\begin{array}{l} E \rightarrow I \mid E + E \mid E * E \mid (E), \\ I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1. \end{array} \quad \longrightarrow \quad \begin{array}{l} E \rightarrow E + T \mid E * T \mid T \\ T \rightarrow (E) \mid I \end{array}$$

$$\begin{array}{l} E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b, \\ E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b. \end{array}$$

# Неоднозначность, левый и правый выводы

- Если существует два различных дерева разбора, то они должны породить два различных левых вывода в соответствии с данным в доказательстве построении.
- Обратно, два различных левых вывода дают различные деревья разбора в соответствии с другой частью доказательства.
- То же самое верно и для правых выводов.

# Неоднозначность, – (2)

- Таким образом, эквивалентным определением для “неоднозначной грамматики” служит:
  1. Существует строка языка, которая имеет два различных левых вывода.
  2. Существует строка языка, которая имеет два различных правых вывода.

# Неоднозначность – это свойство грамматики, а не языка

- Для языка сбалансированных скобок, есть другая КС-грамматика, которая не является неоднозначной.

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow ) \mid (RR$

← B, начальный символ,  
выводит сбалансированные строки.

← R генерирует все сбалансированные строки, за исключением тех, которые имеют правых скобок на одну больше, чем левых. Например:  $)$ ,  $(())$ ,  $((()()))$

Однако, существуют неоднозначные грамматики, для которых нет эквивалентных однозначных.

## Пример: однозначные грамматики

$B \rightarrow (RB \mid \epsilon$        $R \rightarrow ) \mid (RR$

- Построим уникальный левый вывод для данной сбалансированной по скобкам строки путем сканирования строки слева направо.
  - Если нам нужно раскрыть  $B$ , то мы используем  $B \rightarrow (RB$  если следующий символ есть "("; и используем  $\epsilon$  если это конец.
  - Если нам нужно раскрыть  $R$ , используем  $R \rightarrow )$  если следующий символ есть ")" и  $(RR$  если это "(".

# Процесс разбора

Оставшийся вход:

(( ))()



Следующий символ

Шаги левого вывода:

B

$B \rightarrow \underline{RB} \mid \epsilon$

$R \rightarrow ) \mid (RR$



# Процесс разбора

Оставшийся вход:

**( ) ( )**



Следующий символ

Шаги левого вывода

B

(RB

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow ) \mid \underline{(RR}$

# Процесс разбора

Оставшийся вход:

))()



Следующий символ

Шаги левого вывода

B

(RB

((RRB

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow \underline{)} \mid (RR$

# Процесс разбора

Оставшийся вход:

)()



Следующий символ

Шаги левого вывода

B

(RB

((RRB

(()RB

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow \underline{)} \mid (RR$

# Процесс разбора

Оставшийся вход:

()



Следующий символ

Шаги левого вывода:

B

(RB

((RRB

((())RB

((()))B

$B \rightarrow \underline{RB} \mid \epsilon$

$R \rightarrow ) \mid (RR$

# Процесс разбора

Оставшийся вход:

)



Следующий символ

Шаги левого вывода:

B                    (( ))(RB

(RB

((RRB

(( ))RB

(( ))B

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow \underline{)} \mid (RR$

# Процесс разбора

Оставшийся вход:



Следующий символ

Шаги левого вывода:

B            (( ))(RB

(RB        (( ))( )B

((RRB

(( )RB

(( ))B

$B \rightarrow (RB \mid \underline{\epsilon}$

$R \rightarrow ) \mid (RR$

# Процесс разбора

Оставшийся вход:



Следующий символ

Шаги левого вывода:

B            (( ))(RB

(RB        (( ))( )B

((RRB      (( ))( )

(( ))RB

(( ))B

$B \rightarrow (RB \mid \epsilon$

$R \rightarrow ) \mid (RR$

# LL(1) - грамматики

- К слову, такие грамматики как
- $B \rightarrow (RB \mid \epsilon \quad R \rightarrow ) \mid (RR,$
- где вы всегда можете определить правило для использования в левом выводе, сканируя заданную строку слева направо и глядя только на следующий символ, называются LL(1) - грамматики.
  - “Leftmost derivation, left-to-right scan, **one** symbol of lookahead.”



# LL(1) - грамматики – (2)

- Большинство языков программирования имеют LL(1) грамматики.
- LL(1) грамматики никогда не бывают неоднозначными.

# Природная неоднозначность

- Было бы здорово, если бы для каждой неоднозначной грамматики существовал способ «исправить» ее неоднозначность, как это было сделано для грамматики сбалансированных скобок.
- К сожалению, некоторым КС-языкам *по своей природе присуща неоднозначность*, что означает неоднозначность всех грамматик для данного языка.
- Не существует алгоритма, способного определить, является ли произвольная КС-грамматика неоднозначной.

## Пример: Природная неоднозначность

- Язык  $\{0^i 1^j 2^k \mid i = j \text{ или } j = k\}$  неоднозначный по своей природе.
- **Интуитивно**, по крайней мере, некоторые из строк вида  $0^n 1^n 2^n$  должны генерироваться двумя разными деревьями разбора, одним, базирующимся на выборе 0-ей и 1-ц, и другим, базирующимся на выборе 1-й и 2-ек.

# Одна из возможных неоднозначных грамматик

$S \rightarrow AB \mid CD$

A генерирует равное число 0 и 1

$A \rightarrow 0A1 \mid 01$

B генерирует произвольное число 2-ек

$B \rightarrow 2B \mid 2$

$C \rightarrow 0C \mid 0$

C генерирует произвольное число 0-ей

$D \rightarrow 1D2 \mid 12$

D генерирует равное число 1 и 2

Существует два вывода каждой строки

С равным числом 0, 1, и 2. Например:

$S \Rightarrow AB \Rightarrow 01B \Rightarrow 012$

$S \Rightarrow CD \Rightarrow 0D \Rightarrow 012$

# Существенно неоднозначный язык

- Опр. КС-язык  $L$  называется *существенно неоднозначным*, если все его грамматики неоднозначны.
- Если хотя бы одна грамматика языка  $L$  является однозначной, то  $L$  является *однозначным* языком.

# Однозначность грамматик типа 3.

- Теорема. Все языки типа 3 однозначны.
- Доказательство. Пусть  $L$  – язык типа 3 над алфавитом  $\Sigma$ .  
Тогда существует КДА  $M=(Q, \Sigma, \delta, q_0, F)$ , такой что  $L(M)=L$ .  
Но тогда  $L$  порождается грамматикой  $G=(Q, \Sigma, P, q_0)$ , где  
 $P=\{q \rightarrow ar \mid q, r \in Q, a \in \Sigma, \delta(q, a)=r\} \cup \{q \rightarrow \varepsilon \mid q \in F\}$ .  
Очевидно, что  $G$  – однозначная.

# Нормальные формы КС-грамматик

Удаление бесполезных переменных

Удаление  $\varepsilon$ -правил

Удаление единичных правил

Нормальная форма Хомского

# Классификация грамматик и языков по Хомскому: Тип 2

**Опр.** Грамматика  $G = \langle T, N, P, S \rangle$  называется **контекстно-свободной** (КС), если каждое правило из  $P$  имеет вид  $A \rightarrow \beta$ , где  $A \in N$ ,  $\beta \in (T \cup N)^*$ .

В КС-грамматиках допускаются правила с пустыми правыми частями.

Язык, порождаемый контекстно-свободной грамматикой, называется **контекстно-свободным** языком.



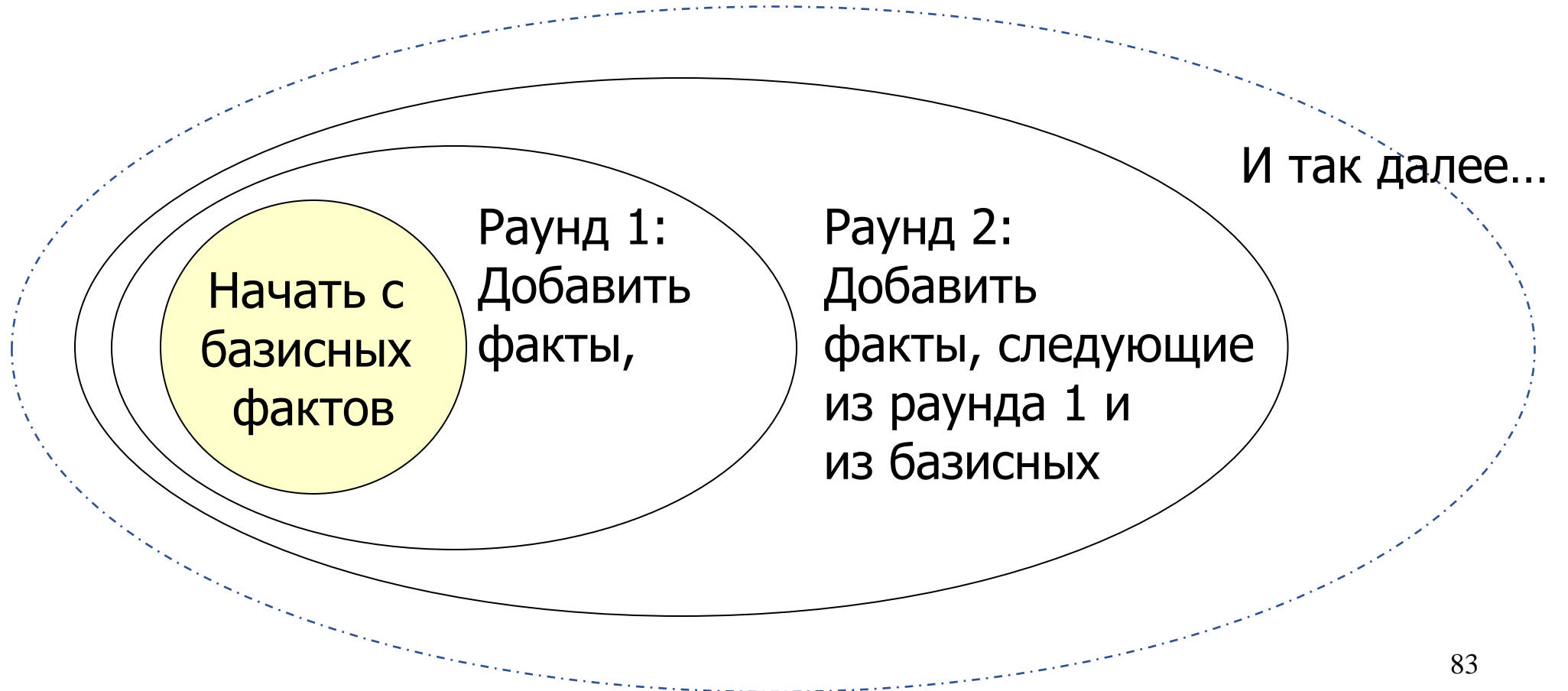
# Переменные, из которых не выводится ничего

- Рассмотрим грамматику с правилами:
  - $S \rightarrow AB$ ,
  - $A \rightarrow aA \mid a$ ,
  - $B \rightarrow AB$
- Хотя  $A$  выводит все строки над алфавитом  $\{a\}$ ,  $B$  не выводит терминальных строк.
  - Почему? Единственная продукция для  $B$  оставляет  $B$  в сентенциальной форме.
- Таким образом,  $S$  не выводит ничего, и соответствующий язык пуст.

# *Упрощающие* алгоритмы

- Есть целое семейство алгоритмов, которые работают по индуктивному принципу.
- Они начинают с открытия (констатации) некоторых очевидных фактов, (базис).
- Они открывают больше фактов из тех, которые уже открыты (индукция).
- В конце концов, когда ничего больше не может быть обнаружено, мы заканчиваем.

# Иллюстрация процесса открытия

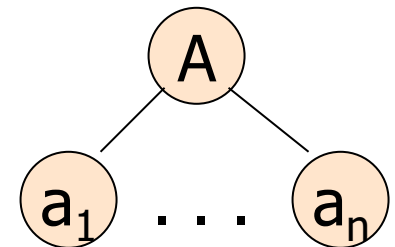


## Проверка выводимости некоторой терминальной строки из переменной

- **Базис:** Если есть правило  $A \Rightarrow w$ , где  $w$  не имеет переменных, то  $A$  выводит терминальную строку (из нетерминального символа  $A$  выводится терминальная цепочка символов).
- **Индукция:** Если есть правило  $A \Rightarrow \alpha$ , где  $\alpha$  состоит только из терминалов и переменных, о которых известно, что они выводят терминальные строки. Тогда  $A$  выводит терминальную строку.
- Мы оканчиваем процесс, когда не можем найти новых переменных.
- Простая индукция по порядку, в котором находятся переменные, показывает, что каждая из них действительно выводит терминальную строку.
- Обратно, любая переменная, которая выводит терминальную строку, может быть найдена приведенным алгоритмом.

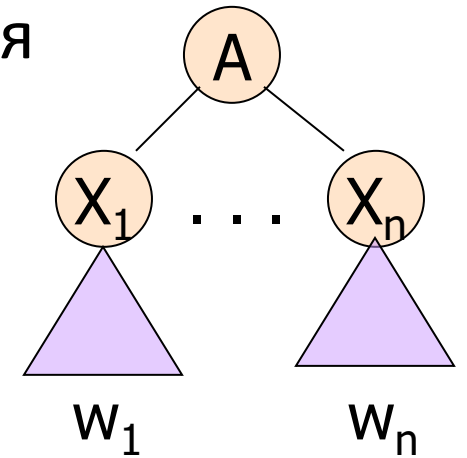
# Доказательство обратного

- Любая переменная, из которой выводится терминальная строка, может быть найдена приведенным алгоритмом.
- Доказательство представляет собой индукцию по высоте дерева синтаксического анализа наименьшей высоты, с помощью которого терминальная последовательность выводится из переменной  $A$ .
- **Базис**: Высота = 1. Дерево выглядит примерно так:  
Базис алгоритма говорит нам, что  $A$  будет найдена.



# Индукция для док-ва обратного

- Пусть, ИН выполняется для деревьев синтаксического анализа высоты  $< h$ , и предположим, что из  $A$  выводится терминальная строка посредством дерева разбора высоты  $h$ :
- По ИН,  $X_i$ 'ы – это те переменные, которые были найдены ранее.
- Тогда,  $A$  также будет найдена, т.к. она имеет справа терминалы и/или уже найденные переменные.



# Алгоритм удаления переменных, из которых ничего не выводится

1. Найдём все переменные, из которых выводятся терминальные строки. Назовем такие переменные *полезными*.
2. Для всех других переменных (*бесполезных*), удалим из грамматики все правила, в которых они появляются либо в голове, либо в теле.

## Пример: Удаление переменных

$S \rightarrow AB \mid C,$      $A \rightarrow aA \mid a,$      $B \rightarrow bB,$      $C \rightarrow c$

- **Базис:** A и C находятся, т.к.  $A \rightarrow a$  и  $C \rightarrow c$ .
- **Индукция:** S находится (помечается), т.к.  $S \rightarrow C$ .
- Больше ничего не находится.
- **Результат:**  $S \rightarrow C,$
- $A \rightarrow aA \mid a,$
- $C \rightarrow c$



# Недостижимые символы

- Другой случай кандидата на удаление терминала или переменной - если она **не может** появиться в каком-либо выводе из начального символа.
- **Базис**: Мы всегда можем достичь  $S$  (начальный символ).
- **Индукция**: Если мы можем достичь  $A$ , и есть правило  $A \Rightarrow \alpha$ , то мы можем достичь  $\alpha$ .

# Недостижимые символы– (2)

- Простые индукции в обоих направлениях показывают, что когда мы не можем обнаружить больше символов, то у нас есть все символы, которые появляются в выводах из  $S$  и только они.
- Идея алгоритма:
- Удалить из грамматики все недостижимые из  $S$  символы и все правила, включающие эти символы.

# Устранение недостижимых символов

**Алгоритм УНС.** Устранение недостижимых символов.

Вход: КС-грамматика  $G = (T, N, P, S)$

Выход: КС-грамматика  $G' = (T', N', P', S)$ , у которой

1.  $L(G') = L(G)$
2. Для всех  $X \in N' \cup \Sigma'$  существуют такие цепочки  $\alpha$  и  $\beta$  из  $(N' \cup \Sigma')^*$ , что  $S \Rightarrow^* \alpha X \beta$  в грамматике  $G'$ .

Метод.

1. Положить  $V_0 = \{S\}$ , и  $i = 1$ .
2. Положить  $V_i = \{X \mid \text{в } P \text{ есть } A \rightarrow \alpha X \beta \text{ и } A \in V_{i-1}\} \cup V_{i-1}$ .
3. Если  $V_i \neq V_{i-1}$ , положить  $i = i + 1$  и перейти к шагу 2, иначе:  
 $N' = V_i \cap N$ ,  $T' = V_i \cap T$ ,  
 $P'$  состоит из правил множества  $P$ , содержащих только символы из  $V_i$ ,  
 $G' = (T', N', P', S)$

# Удаление бесполезных символов

- Опр. Символ называется *полезным*, если он появляется в некотором выводе какой-либо терминальной строки из начального символа.
- В противном случае, он является *бесполезным*.
- Удалим все бесполезные символы:
- Идея алгоритма:
  - 1) Удалим все символы, из которых не выводятся терминальные строки.
  - 2) Удалим недостижимые символы.

## Пример: Бесполезные символы – (2)

$S \rightarrow AB$ ,  $A \rightarrow C$ ,  $C \rightarrow c$ ,  $B \rightarrow bB$

- Как и следует, мы сначала удаляем символы, из которых не выводятся терминальные строки и соответствующие правила, т.е. удаляем  $B$ .
- Но, удалив  $B$ , мы удаляем правило  $B \rightarrow bB$ , а затем и правило вывода из начального символа  $S \rightarrow AB$ .
- Затем, мы применяем алгоритм поиска недостижимых символов из начального и видим, что всё недостижимо, т.е. все правила будут удалены.
- Однако, если мы сначала будем удалять недостижимые символы, то увидим, что всё достижимо из  $S$ , т.е. ничего не удалиться.
- Т.о.  $A$ ,  $C$ , и  $c$  никогда не будут удалены, останутся правила  $A \rightarrow C$ ,  $C \rightarrow c$ , которых не должно быть, т.к.  $A$ ,  $C$ , и  $c$  – бесполезные.

# Разрешимость проблемы пустоты КС-грамматик

**Алгоритм.** Не пуст ли язык  $L(G)$ ?

**Вход:** КС-грамматика  $G = (T, N, P, S)$

**Выход:** «ДА», если  $L(G) \neq \emptyset$  и «НЕТ» – в противном случае.

**Метод.** Рекурсивно строим множества  $N_0, N_1, \dots$

1.  $N_0 = \emptyset, i = 1$
2.  $N_i = \{A \mid A \rightarrow \alpha \in P \text{ и } \alpha \in (N_{i-1} \cup T)^*\} \cup N_{i-1}$
3. Если  $N_i \neq N_{i-1}$ , то  $i = i + 1$  и перейти к 2, иначе  $N_e = N_i$
4. Если  $S \in N_e$ , то результат «ДА», иначе - «НЕТ».

• **Следствие.**

Для КС-грамматики  $G$  проблема пустоты языка  $L(G)$  **разрешима**.

# Устранение бесполезных символов

**Алгоритм УБС.** Устранение бесполезных символов

Вход: КС-грамматика  $G = (T, N, P, S)$ , у которой  $L(G) \neq \emptyset$ .

Выход: КС-грамматика  $G' = (T', N', P', S)$ , у которой  $L(G') = L(G)$  и в  $N' \cup T'$  нет бесполезных символов.

Метод.

1. Применив к  $G$  алгоритм определения непустоты языка, получить  $N_e$ .

Положить  $G_1 = \langle T, N \cap N_e, P_1, S \rangle$ , где  $P_1$  состоит из правил мн-ва  $P$ , содержащих только символы из  $N_e \cap T$ .

2. Применив к  $G_1$  алгоритм устранения недостижимых символов, получить  $G' = (T', N', P', S)$

# Почему это работает

- **Теорема.** Грамматика  $G'$ , которую строит алгоритм «УБС» не содержит бесполезных символов и  $L(G') = L(G)$ .
- **Доказательство.**
- В части (1)  $N_e$  строиться за конечное число шагов.  
После шага (1), каждый оставшийся символ выводит некоторую терминальную строку.
- Грамматика в части (2) также строиться за конечное число шагов.  
После шага (2) только все оставшиеся символы выводимы из  $S$ .  
К тому же, они ещё выводят терминальные строки, т.к. **такие** выводы могут включать только символы, выводимые из  $S$ .



# $\varepsilon$ -правила

- **Опр.** КС-грамматика  $G = \langle T, N, P, S \rangle$  называется грамматикой без  $\varepsilon$ -правил, если в ней нет  $\varepsilon$ -правил, за исключением, может быть, правила  $S \rightarrow \varepsilon$  и если  $S$  не встречается в правых частях правил.

# $\epsilon$ -правила

- Мы можем почти избежать правил вида  $A \rightarrow \epsilon$  (называемых  *$\epsilon$ -правилами* ).
  - Проблема в том, что  $\epsilon$  не может быть в языке какой-либо грамматики, если в ней нет  $\epsilon$ -правил.
- **Теорема**: Если  $L$  — КС-язык, то язык  $L - \{\epsilon\}$  имеет КС-грамматику без  $\epsilon$ -правил.

# Пустые символы

- Чтобы удалить  $\epsilon$ -правила, нужно сначала найти *пустые символы* = переменные  $A$  такие, что  $A \Rightarrow^* \epsilon$ .
- **Базис**: Если есть правило  $A \rightarrow \epsilon$ , то  $A$  пустой.
- **Индукция**: Если есть правило  $A \rightarrow \alpha$ , и все символы в  $\alpha$  пустые, то  $A$  пустой.

## Пример: пустые символы

$S \rightarrow AB,$

$A \rightarrow aA \mid \epsilon,$

$B \rightarrow bB \mid A$

- **Базис:**  $A$  – пустой, т.к.  $A \rightarrow \epsilon$ .
- **Индукция:**  $B$  – пустой, т.к.  $B \rightarrow A$ .
- Далее,  $S$  – пустой, т.к.  $S \rightarrow AB$ .

# Удаление $\epsilon$ -правил

- **Идея:** преобразовать каждое правило  $A \rightarrow X_1 \dots X_n$  в семейство правил.
- Для каждого подмножества пустых  $X$ -ов в теле правила, формируем одно новое правило с удалением этих  $X$ -ов в правой части исходного правила».
- **Исключение!** Если все  $X$ -ы пустые (или тело правила изначально было пустым), то не образуем правило с правой частью  $\epsilon$ .

# Удаление $\epsilon$ -правил

**Алгоритм УЭП.** Преобразование в грамматику без  $\epsilon$ -правил.

Вход: КС-грамматика  $G = (T, N, P, S)$

Выход: КС-грамматика  $G' = (T', N', P', S')$  без  $\epsilon$ -правил.

Метод.

1. Построить  $N_e = \{A \mid A \in N \text{ и } A \Rightarrow_G^+ \epsilon\}$

2. Построить  $P'$ :

- Если  $A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \dots B_k \alpha_k$  принадлежит  $P$ ,  $k \geq 0$  и  $B_i \in N_e$  для  $1 \leq i \leq k$ , но ни один символ в цепочках  $\alpha_i$  ( $0 \leq i \leq k$ ) не принадлежит  $N_e$ , то включить в  $P'$  все правила вида

$A \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \dots \alpha_{k-1} X_k \alpha_k$ , где  $X_i$  либо  $B_i$ , либо  $\epsilon$  (но не включать правило  $A \rightarrow \epsilon$  в случае, когда все  $\alpha_i = \epsilon$ ).

- Если  $S \in N_e$ , включить в  $P'$  правила  $S' \rightarrow \epsilon \mid S$ , где  $S'$  – новый нетерминал и положить  $N' = N \cup \{S'\}$ . В противном случае положить  $N' = N$  и  $S' = S$ .

3. Определить  $G' = (T', N', P', S')$ .

# Пример

- Правила:  $S \rightarrow aSbS \mid bSaS \mid \varepsilon$
- После применения алгоритма получим правила:
- $S' \rightarrow S \mid \varepsilon$
- $S \rightarrow aSbS \mid bSaS \mid aSb \mid abS \mid bSa \mid ab \mid baS \mid ba$

## Пример: Удаление $\epsilon$ -правил

$S \rightarrow ABC, \quad A \rightarrow aA \mid \epsilon, \quad B \rightarrow bB \mid \epsilon, \quad C \rightarrow \epsilon$

- $A, B, C$ , и  $S$  пустые.
- Новая грамматика:

$S \rightarrow \cancel{ABC} \mid AB \mid \cancel{AC} \mid \cancel{BC} \mid A \mid B \mid \cancel{C}$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

**Замечание:**  $C$  - теперь бесполезный.  
Удаляем его правила.



# Почему это работает

- **Теорема.** Алгоритм «УЄП» даёт грамматику без  $\epsilon$ -правил, эквивалентную входной грамматике.
- **Докажем**, что для всех переменных  $A$ :
  - 1) Если  $w \neq \epsilon$  и  $A \Rightarrow_{\text{old}}^* w$ , то  $A \Rightarrow_{\text{new}}^* w$ .
  - 2) Если  $A \Rightarrow_{\text{new}}^* w$  то  $w \neq \epsilon$  и  $A \Rightarrow_{\text{old}}^* w$ .
- Далее, полагая  $A$  в качестве начального символа, докажем, что  $L(\text{new}) = L(\text{old}) - \{\epsilon\}$ .
- (1) – индукция по числу шагов, в результате которых из  $A$  выводится  $w$  в старой грамматике

## Док-во 1 – Базис

- Если старый вывод делается за один шаг, то  $A \rightarrow w$  должно быть правилом.
- Т.к.  $w \neq \epsilon$ , это правило также окажется в новой грамматике.
- Таким образом,  $A \Rightarrow_{\text{new}} w$ .

## Док-во 1 – Индукция

- Пусть  $A \Rightarrow_{old}^* w$  есть вывод за  $k$  шагов, и предположим, что ИН выполняется для выводов длиной меньше, чем  $k$ .
- Пусть первым шагом вывода будет  $A \Rightarrow_{old} X_1 \dots X_n$ .
- Тогда  $w$  можно разбить:  $w = w_1 \dots w_n$ , где для всех  $i$ ,  $w_i$  - часть  $w$ , которая либо есть  $X_i$  (если  $X_i$ - терминал), либо  $X_i \Rightarrow_{old}^* w_i$ , за меньшее, чем  $k$  шагов.

## Индукция – Продолжение

- Если  $X_i$ - переменная и  $w_i \neq \epsilon$ , то по IH,  $X_i \Rightarrow_{new}^* w_i$ .
- Также, новая грамматика имеет правило с  $A$  в левой части, и с  $X_i$ -ами в правой, для которых  $w_i \neq \epsilon$ .
  - **Заметим**: они не могут все быть  $\epsilon$ , потому что  $w \neq \epsilon$ .
- При использовании этого правила в новой грамматике по выводам  $X_i \Rightarrow_{new}^* w_i$ , имеем, что из  $A$  выводится  $w$  в новой грамматике.

## Доказательство части (2)

- Мы также должны доказать часть (2) – если  $w$  выводится из  $A$  в новой грамматике, то она непустая и выводится также в старой грамматике.
- Доказательство проводится по индукции аналогично.

# Единичные (цепные) правила

- **Опр.** *Единичные правила* - это те правила, в теле которых есть только одна переменная.
- Эти правила могут быть удалены.
- **Ключевая идея:** Если  $A \Rightarrow^* B$  в результате серии единичных выводов, и  $B \rightarrow \alpha$  неединичное правило, то добавим правило  $A \rightarrow \alpha$ .
- Затем, удалим все единичные правила.

## Единичные правила – (2)

- Алгоритм.
- Найдем все пары  $(A, B)$  такие, что  $A \Rightarrow^* B$  только последовательностью единичных правил.
- **Базис:** Очевидно  $(A, A)$  - переменная выводится сама из себя за 0 шагов.
- **Индукция:** Если мы уже нашли  $(A, B)$ , и  $B \rightarrow C$  – единичное правило, то добавим  $(A, C)$ .

## Доказательство, что мы нашли точно только правильные пары

- Индукцией по порядку, в котором пары  $(A, B)$  найдены, можно показать, что  $A \Rightarrow^* B$  по единичным правилам.
- Обратно, индукцией по числу шагов в выводе единичными правилами  $A \Rightarrow^* B$ , мы можем показать, что пара  $(A, B)$  найдена.



# Алгоритм устранения единичных правил

- Алгоритм УЕП.
- Вход. КС-грамматика без  $\varepsilon$ -правил.
- Выход. Эквивалентная КС-грамматика  $G'$  без  $\varepsilon$ -правили без единичных правил.
- Метод. 1. Для каждого  $A \in N$  построить  $N_A = \{B | A \Rightarrow^* B\}$  следующим образом:
  - (а) Положить  $N_0 = \{A\}$  и  $i=1$ .
  - (б) Положить  $N_i = \{C | B \rightarrow C \in P \text{ и } B \in N_{i-1}\} \cup N_{i-1}$ .
  - (в) Если  $N_i \neq N_{i-1}$ , то положить  $i=i+1$  и повторить шаг (б), иначе  $N_A = N_i$ .
- 2. Построить  $P'$ : если  $B \rightarrow \alpha \in P$  и не является единичным правилом, включить в  $P'$  правило  $A \rightarrow \alpha$  для всех таких  $A$ , что  $B \in N_A$ .
- 3. Положить  $G' = (N, \Sigma, P', S)$ .

# Доказательство, что алгоритм удаления единичных правил работает

- Теорема. Алгоритм УЕП строит грамматику  $G'$  без единичных правил и  $L(G')=L(G)$ .
- **Основная идея**: Вывод  $A \Rightarrow_{lm}^* w$  существует в новой грамматике т.и.т.т., когда такой вывод есть в старой.
- Последовательность единичных правил и неединичное правило свёртываются в одно правило новой грамматики.

# Очистка грамматики

- Теорема: Если  $L$  есть КС-язык, то существует КС-грамматика для языка  $L - \{\epsilon\}$  такая, что в ней:
  1. Нет бесполезных символов.
  2. Нет  $\epsilon$ -правил.
  3. Нет единичных правил.
- Т.е., каждое тело правил есть либо единичный терминал, либо имеет длину  $\geq 2$ .
- Такого рода КС-грамматика называется «приведенной».

# Очистка грамматики– (2)

■ Док-во: Начнём с КС-грамматики для L.

■ Выполним следующие шаги:

1. Удалим  $\epsilon$ -правила.
2. Удалим единичные правила.
3. Удалим переменные, из которых не выводятся терминальные строки.
4. Удалим переменные, которые недостижимы из начального символа.

Д.б. первым.  
Может породить  
единичные правила  
или бесполезные  
переменные.

# Нормальная форма Хомского

- **Опр.** Говорят, что КС-грамматика находится в *нормальной форме Хомского*, если каждое ее правило имеет один из следующих видов:
  1.  $A \rightarrow BC$  (тело имеет две переменные).
  2.  $A \rightarrow a$  (тело – единичный терминал).
- **Теорема:** Если  $L$  – КС-язык, то  $L - \{\epsilon\}$  имеет КС-грамматику в НФХ.

# Доказательство теоремы о НФХ

- **Шаг 1:** “Очистим” грамматику, так, что тело каждого правила есть либо один терминал, либо имеет длину, по крайней мере, 2.
- **Шаг 2:** Для каждого тела правила  $\neq$  единичный терминал, создаем новые переменные для правой стороны правила.
  - Для каждого терминала  $a$  создаём новую переменную  $A_a$  и правило  $A_a \rightarrow a$ .
  - Заменяем  $a$  на  $A_a$  в теле правил длиной  $\geq 2$ .

## Пример: Шаг 2

- Рассмотрим правило  $A \rightarrow BcDe$ .
- Нам нужны переменные  $A_c$  и  $A_e$ . с правилами  $A_c \rightarrow c$  и  $A_e \rightarrow e$ .
  - **Заметим**: мы создаем по крайней мере одну переменную для каждого терминала, и используем ее везде, где нужно.
- Заменяем  $A \rightarrow BcDe$  на  $A \rightarrow BA_cDA_e$ .

## НФХ: Доказательство – продолжение

- Шаг 3: Разобьём правые части, длина которых больше 2, на цепочку правил с правыми частями из двух переменных.
- Пример:  $A \rightarrow BCDE$  заменяется на
$$A \rightarrow BF, F \rightarrow CG, \text{ и } G \rightarrow DE.$$
  - F и G должны использоваться везде.



## Пример шага 3 – продолжение

- Итак,  $A \rightarrow BCDE$  заменяется на  $A \rightarrow BF$ ,  $F \rightarrow CG$ , и  $G \rightarrow DE$ .
- В новой грамматике,  $A \Rightarrow BF \Rightarrow BCG \Rightarrow BCDE$ .
- **Более важно:** Однажды заменив  $A$  на  $BF$ , мы продолжим выводить  $BCG$  и  $BCDE$ .
  - Т.к.  $F$  и  $G$  имеют только одно правило.

Преобразовать в нормальную форму  
Хомского КС-грамматику  $G = (N, \Sigma, P, S)$

$$S \rightarrow SS, S \rightarrow 1A0, A \rightarrow 1A0, A \rightarrow \varepsilon$$

$S \rightarrow SS | 1A0$   
 $A \rightarrow 1A0$   
 $A \rightarrow \varepsilon$

удаляем  $\varepsilon$  правила  
 $S \rightarrow SS | 1A0 | 10$   
 $A \rightarrow 1A0 | 10$

цепные:  
нет таких

бесполезные:  
нет таких



$S \rightarrow SS | EAO | EO$   
 $E \rightarrow 1$   
 $O \rightarrow 0$   
 $A \rightarrow EAO | EO$



$S \rightarrow SS | ER | EO$   
 $R \rightarrow AO$   
 $E \rightarrow 1$   
 $O \rightarrow 0$   
 $A \rightarrow ER | EO$

Преобразовать в нормальную форму Хомского КС-грамматику

$$G = (N, \Sigma, P, S)$$

$$\underline{S} \rightarrow \underline{AB}, A \rightarrow SA, \underline{A} \rightarrow \underline{BB}, A \rightarrow bB, B \rightarrow b, B \rightarrow aA, \underline{B} \rightarrow \epsilon$$

$S \rightarrow AB$

$A \rightarrow SA | BB | bB$

$B \rightarrow b | aA | \epsilon$

1. удаляем  $\epsilon$ -правила
2. удаляем цепные (единичные) правила
3. удаляем бесполезные символы

1. удаляем  $\epsilon$ -правила

$S \rightarrow A | AB | B$

$A \rightarrow SA | BB | B | bB | b | \underline{S} | \underline{A}$

$B \rightarrow b | aA | a$

2. удаляем цепные правила

$(S, S) \& S \rightarrow A \quad (S, A)$

$(S, A) \& A \rightarrow B \quad (S, B)$

$(A, A) \& A \rightarrow B \quad (A, B)$

$(S, S) \quad S \rightarrow AB$

$(S, A) \quad S \rightarrow SA | BB | bB | b$

$(S, B) \quad S \rightarrow b | aA$

$S \rightarrow AB | SA | BB | bB | b | aA | a$

$A \rightarrow SA | BB | bB | b | aA | a$

$B \rightarrow b | aA | a$

3. удаляем бесполезные  
таких нет

Приведение к НФХ

$S \rightarrow AB | SA | BB | \underline{KB} | b | \underline{RA} | a$

$K \rightarrow b$

$R \rightarrow a$

$A \rightarrow SA | BB | \underline{KB} | b | \underline{RA} | a$

$B \rightarrow b | \underline{RA} | a$

# Автоматы с магазинной памятью

Определение

Функционирование МП-автомата

Языки для МП-автоматов

Детерминированные МП-автоматы

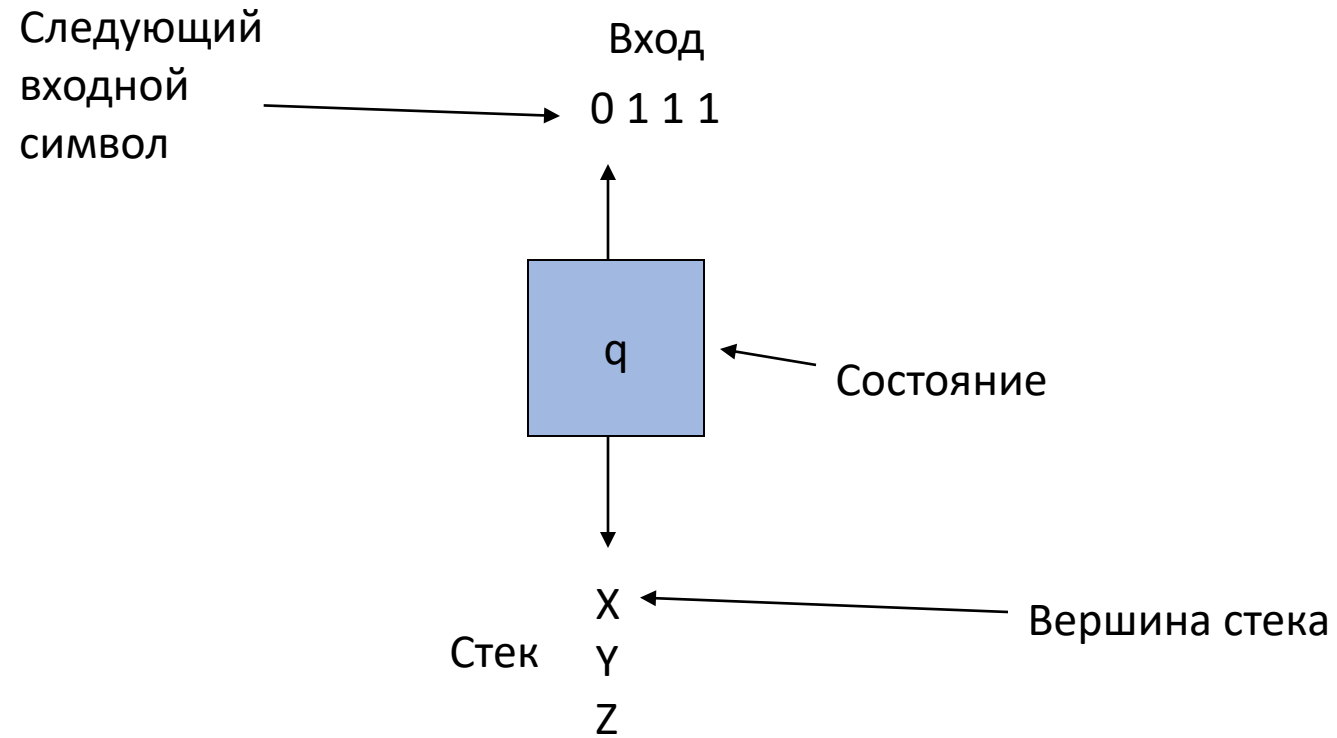
# Автоматы с магазинной памятью

- МП-автомат – это автомат эквивалентный по своей выразительной мощности определения КС-языков.
- Только недетерминированные МП-автоматы определяют все КС-языки.
- Но детерминированная версия моделирует синтаксические анализаторы.
  - Большинство языков программирования имеют детерминированные МП-автоматы.

# Неформально: МПА

- Вообразим  $\epsilon$ -НКА с дополнительной возможностью манипулирования со стеком.
- Функционирование МПА определяется следующими параметрами:
  1. Текущее состояния (его “НКА”),
  2. Текущий входной символ (или  $\epsilon$ ), и
  3. Текущий символ вершины стека.

# Схема МПА



## Интуитивно: МПА – (2)

- Недетерминированность: МПА может иметь выбор при следующих тактах работы.
- При каждом выборе МПА может:
  1. Изменить состояние, а также
  2. Заменить символ вершины стека последовательностью из 0 или более символов.
    - Нуль символов = “pop.”
    - Много символов = последовательность “pushes.”



# МПА формально

- МПА определяется:  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ 
  1. Конечным множеством *состояний* ( $Q$ , обычно).
  2. *Входным алфавитом* ( $\Sigma$ , обычно).
  3. *Алфавитом стека* ( $\Gamma$ , обычно).
  4. *Функцией переходов* ( $\delta$ , обычно).
  5. *Начальным состоянием* ( $q_0$ , в  $Q$ , обычно).
  6. *Начальным символом* ( $Z_0$ , в  $\Gamma$ , обычно).
  7. Множеством *конечных состояний* ( $F \subseteq Q$ , обычно).

# Соглашения

- $a, b, \dots$  - входные символы.
  - Но иногда мы позволяем использовать  $\epsilon$  в качестве возможного значения.
- $\dots, X, Y, Z$  – символы стека.
- $\dots, w, x, y, z$  – строки входных символов
- $\alpha, \beta, \dots$  - строки стековых символов.

# Функция переходов

- Имеет три аргумента:
  1. Состояние, в  $Q$ .
  2. Вход, который является либо символом в  $\Sigma$  или  $\epsilon$ .
  3. Верхний стековый символ в  $\Gamma$ .
- $\delta(q, a, Z)$  – есть множество нуль или более действий вида  $(p, \alpha)$ .
  - $p$  – новое состояние;  $\alpha$  - строка стековых символов, которая заменяет символ вершины стека.

# Действия МПА

- Если  $\delta(q, a, Z)$  содержит  $(p, \alpha)$  среди его действий, то единственным, что МПА может сделать в состоянии  $q$ , с  $a$  в начале входа, и  $Z$  в вершине стека, это:
  1. Изменить состояние на  $p$ .
  2. Удалить символ  $a$  из начальной части входа (но  $a$  может быть  $\epsilon$ ).
  3. Заменить  $Z$  в вершине стека на  $\alpha$ .
- МПА может иметь несколько альтернативных пар вида  $(p, \alpha)$  для  $\delta(q, a, Z)$ .

## Пример: МПА

- Построим МПА, принимающий  $\{0^n 1^n \mid n \geq 1\}$ .
- Состояния:
  - $q$  = начальное состояние. Мы находимся в состоянии  $q$ , если мы видели до данного момента только 0'и.
  - $p$  = мы увидели, по крайней мере, одну 1-цу и можем теперь обрабатывать только входы из 1'ц.
  - $f$  = конечное состояние; принятие.

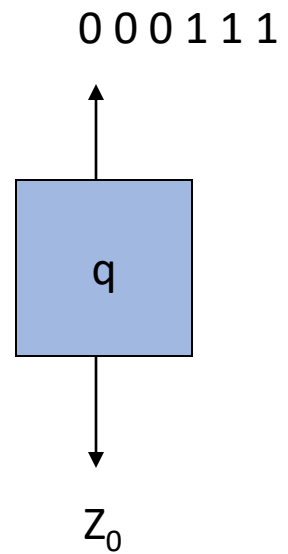
## Пример: МПА – (2)

- Символы стека:
  - $Z_0$  = начальный символ. Помечает также дно стека, так, что мы знаем, что посчитали такое же число 1'ц что и 0'ей.
  - $X$  = маркер, используемый для подсчёта числа 0'ей наблюдаемых на входе.

## Пример: МПА – (3)

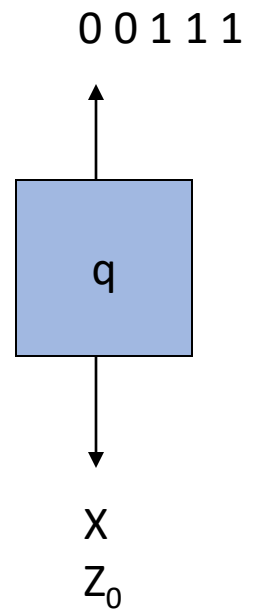
- Переходы:
  - $\delta(q, 0, Z_0) = \{(q, XZ_0)\}$ .
  - $\delta(q, 0, X) = \{(q, XX)\}$ . Данные два правила обеспечивают помещение одного  $X$  в стек для каждого  $0$ , читаемого из входа.
  - $\delta(q, 1, X) = \{(p, \epsilon)\}$ . Когда мы видим  $1$ , переходим в состояние  $p$  и удаляем из стека один символ  $X$ .
  - $\delta(p, 1, X) = \{(p, \epsilon)\}$ . Удаляем одни  $X$  для каждой  $1$ -ы.
  - $\delta(p, \epsilon, Z_0) = \{(f, Z_0)\}$ . Принимаем при достижении дна стека.

# Действия: Пример МПА

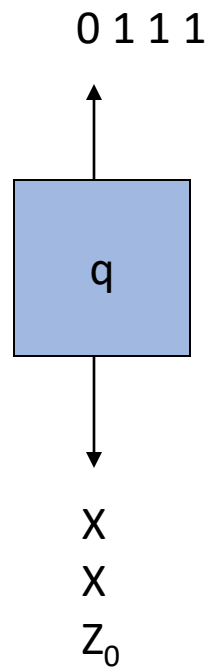




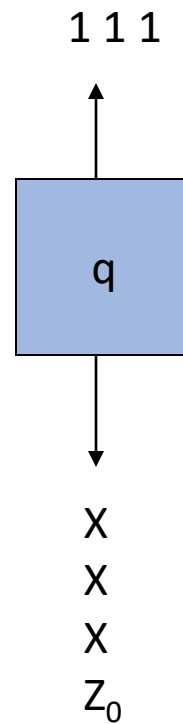
# Действия: Пример МПА



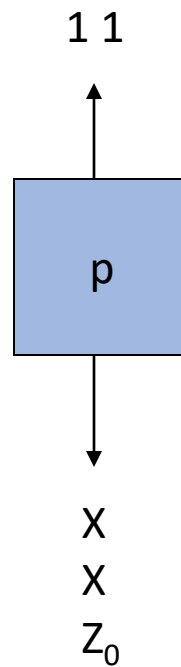
# Действия: Пример МПА



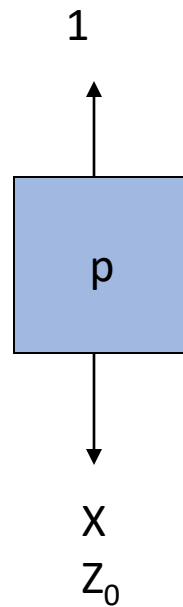
# Действия: Пример МПА



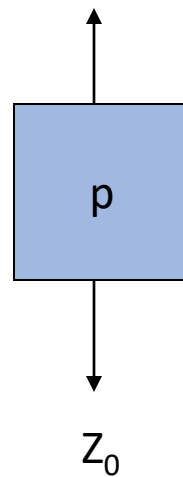
# Действия: Пример МПА



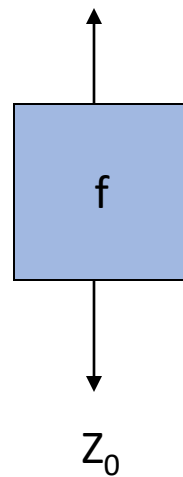
# Действия: Пример МПА



Действия: Пример МПА



Действия: Пример МПА



# Мгновенная конфигурация

- Мы можем формализовать описание работы МПА на основе *мгновенных конфигураций* (ID- instantaneous description).
- ID – это тройка  $(q, w, \alpha)$ , где:
  1.  $q$  – текущее состояние.
  2.  $w$  – оставшийся непрочтенный вход.
  3.  $\alpha$  - содержимое стека, с вершиной слева.



# Отношение смены конфигураций

- Чтобы сказать, что ID I может измениться на ID J при одном шаге работы МПА, мы пишем  $I \vdash J$ .
- Формально,  $(q, aw, X\alpha) \vdash (p, w, \beta\alpha)$  для любого w и  $\alpha$ , если  $\delta(q, a, X)$  содержит  $(p, \beta)$ .
- Расширим  $\vdash$  на  $\vdash^*$ , означающее “нуль или более шагов,” следующим образом:
  - **Базис:**  $I \vdash^* I$ .
  - **Индукция:** If  $I \vdash^* J$  и  $J \vdash K$ , то  $I \vdash^* K$ .

## Пример смены конфигураций

- Используя предыдущий пример МПА, мы можем описать последовательность шагов:
- $(q, 000111, Z_0) \vdash (q, 00111, XZ_0) \vdash (q, 0111, XXZ_0) \vdash$   
 $\vdash (q, 111, XXXZ_0) \vdash (p, 11, XXZ_0) \vdash (p, 1, XZ_0) \vdash (p, \epsilon, Z_0) \vdash (f, \epsilon, Z_0)$
- Таким образом,  $(q, 000111, Z_0) \vdash^* (f, \epsilon, Z_0)$ .
- Что могло бы случиться при входе 0001111?

## Ответ

- $(q, 0001111, Z_0) \vdash (q, 001111, XZ_0) \vdash (q, 01111, XXZ_0) \vdash (q, 1111, XXXZ_0) \vdash (p, 111, XXZ_0) \vdash (p, 11, XZ_0) \vdash (p, 1, Z_0) \vdash (f, 1, Z_0)$
- Заметим, что ID не имеет возможностей изменений.
- 0001111 **не** принимается, т.к. вход не полностью обработан.

# Язык МПА

- Общим способом определения языка, определяемого МПА, является его *конечное состояние*.
- Если  $P$  есть МПА, то  $L(P)$  есть множество строк  $w$  таких, что  $(q_0, w, Z_0) \vdash^* (f, \epsilon, \alpha)$  для конечного состояния  $f$  и любого  $\alpha$ .

## Язык МПА – (2)

- Другим способом определения языка для того же МПА PDA является признак *пустоты стека*.
- Если  $P$  есть МПА, то  $N(P)$  есть множество строк  $w$  таких, что  $(q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)$  для любого состояния  $q$ .

# Эквивалентность определения языков

## Теорема

1. Если  $L = L(P)$ , то существует другой МПА  $P'$  такой, что  $L = N(P')$ .
2. Если  $L = N(P)$ , то существует другой МПА  $P''$  такой, что  $L = L(P'')$ .

## Доказательство: $L(P) \rightarrow N(P')$ интуитивно

- $P'$  моделирует  $P$ .
- Если  $P$  принимает строку,  $P'$  опустошит свой стек.
- $P'$  должен избежать случайного опустошения стека, для этого он использует специальный маркер дна стека на случай, когда  $P$  опустошает свой стек без принятия входа.

## Доказательство: $L(P) \rightarrow N(P')$

- $P'$  имеет все состояния, символы, и переходы, что и у  $P$ , плюс:
  1. Стековый символ  $X_0$  (начальный стековый символ  $P'$ ), используемый для отслеживания дна стека.
  2. Новое начальное состояние  $s$  и “удаляющее” состояние  $e$ .
  3.  $\delta(s, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$ . Начало работы с  $P$ .
  4. Добавим  $\{(e, \epsilon)\}$  к  $\delta(f, \epsilon, X)$  для любого финального состояния  $f$  автомата  $P$  и любого стекового символа  $X$ , включая  $X_0$ .
  5.  $\delta(e, \epsilon, X) = \{(e, \epsilon)\}$  для любого  $X$ .



## Доказательство: $N(P) \rightarrow L(P'')$ Интуитивно

- $P''$  моделирует  $P$ .
- $P''$  имеет специальный маркер дна стека, чтобы отследить ситуацию, когда  $P$  опустошает свой стек.
- Если это так, то,  $P''$  переходит в заключительное состояние и принимает входную строку.

## Доказательство : $N(P) \rightarrow L(P'')$

- $P''$  имеет все состояния, символы и переходы, что и у  $P$ , плюс:
  1. Стековый символ  $X_0$  (начальный символ), используемый для отслеживания дна стека.
  2. Новое начальное состояние  $s$  и финальное состояние  $f$ .
  3.  $\delta(s, \epsilon, X_0) = \{(q_0, Z_0X_0)\}$ . Начало работы с  $P$ .
  4.  $\delta(q, \epsilon, X_0) = \{(f, \epsilon)\}$  для любого состояния  $q$  автомата  $P$ .

# Детерминированные МПА

- Чтобы быть детерминированным, у МПА должен быть, по крайней мере, один выбор при переходе из любого состояния  $q$ , для входного символа  $a$ , и стекового символа  $X$ .
- Дополнительно, не должно быть выбора между использованием входа  $\epsilon$  или реального входа.
  - Формально:  $\delta(q, a, X)$  и  $\delta(q, \epsilon, X)$  не могут быть оба непусты.
- Обычно принятие входа Д-МПА определяется переходом в заключительное состояние, т.к. при пустом стеке мы не можем больше обрабатывать вход
- Хотя мы не будем углубляться далее в теорию МПА, отметим, что класс языков, принятых детерминированными МПА, содержит все регулярные языки (очевидно, поскольку он может моделировать детерминированный конечный автомат, просто игнорируя его стек), но не включает все контекстно-свободные языки.

# Эквивалентность МП-автоматов и КС-грамматик

Преобразование КС-грамматики в  
МПА

Преобразование МПА в КС-  
грамматики

# Обзор

- Когда мы говорили о свойствах замыкания для регулярных языков, было полезно переключаться между представлениями в виде РВ и КДА.
- Подобно этому, оба представления в виде КС-грамматики и МПА являются полезными при определении свойств КС-языков.

## Обзор – (2)

- Кроме того, МПА, которые будучи “алгоритмичными,” зачастую проще использовать, при определении принадлежности языка к множеству КС-языков.
- **Пример:** Легко видеть, как МПА может распознать сбалансированность скобок; в то время, как это не так легко, с помощью КС-грамматики.

# Преобразование КСГ в МПА

- **Теорема**. Пусть  $G$  – КС-грамматика и  $L = L(G)$ .
- Тогда можно построить МПА  $P$  такой, что  $N(P) = L$ .
- $P$  имеет:
  - Одно состояние  $q$ .
  - Входные символы = терминалы грамматики  $G$ .
  - Стековые символы = все символы  $G$ .
  - Начальный символ = начальный символ  $G$ .

# Интуитивно о Р

- На каждом шаге, Р представляет некоторую *лево-сентенциальную форму* (шаг левого вывода) из начального символа S.
- Если стек Р есть  $\alpha$ , и Р только что обработал  $x$  из своего входа, то Р представляет лево-сентенциальную форму  $x\alpha$ .
- При пустом стеке, обработанной строкой является строка в  $L(G)$ .
- Если никакая последовательность вариантов работы недетерминированного МП-автомата Р не приводит к пустому стеку после обработки  $w$  из входа, то  $w$  не является терминальной строкой, порождаемой грамматикой, и Р, соответственно, не принимает  $w$ .



# От МПА к КСГ

- Теперь предположим, что  $L = N(P)$ .
- **Теорема**. Пусть  $P$  – МПА и  $L = N(P)$ . Тогда можно построить КС-грамматику  $G$  такую, что  $L = L(G)$ .
- **Интуитивно**:  $G$  будет иметь переменные  $[pXq]$  генерирующие в точности строки  $w$ , которые являются причиной для  $P$  иметь эффект выталкивания символа  $X$  из стека при переходе из состояния  $p$  в состояние  $q$ .
- При этом  $P$  может наращивать стек значительно выше того, где был  $X$ .
  - При этом  $P$  никогда не опускается ниже этого  $X$ .

# Лемма о накачке для КС-языков

Формулировка

Приложения

# ИНТУИТИВНО

- Вспомним лемму о накачке для регулярных языков.
- Она говорит, что если есть строка достаточной длины, доставляющая цикл в допускающем соответствующий язык КДА, то мы можем продублировать цикл и найти бесконечное число строк из того же языка.

# Интуиция – (2)

- Для КС-языков ситуация немного сложнее.
- Мы можем всегда найти **две** части любой достаточно длинной строки, чтобы, чтобы “раздуть” их в тандеме.
  - **То есть:** если мы повторим каждую из двух частей одно и то же число раз, мы получим другую строку из того же языка.

- Лемма. (О длине выводимой цепочки).
- Пусть дано дерево разбора для грамматики  $G = (T, N, P, S)$  в нормальной форме Хомского. Пусть кроной дерева является цепочка  $\alpha \in \Sigma^+$ .
- Если  $n$  – наибольшая высота пути от корня к листьям (высота дерева), то  $|\alpha| \leq 2^{n-1}$ .



- Доказательство индукцией по высоте дерева.

# Формулировка леммы о накачке для КС-языков

**Лемма.** Для каждого КС-языка  $L$  существует целое число  $n$ , такое, что для каждой строки  $z$  из  $L$  длины  $\geq n$  существует ее представление  $z = uvwx$  такое что:

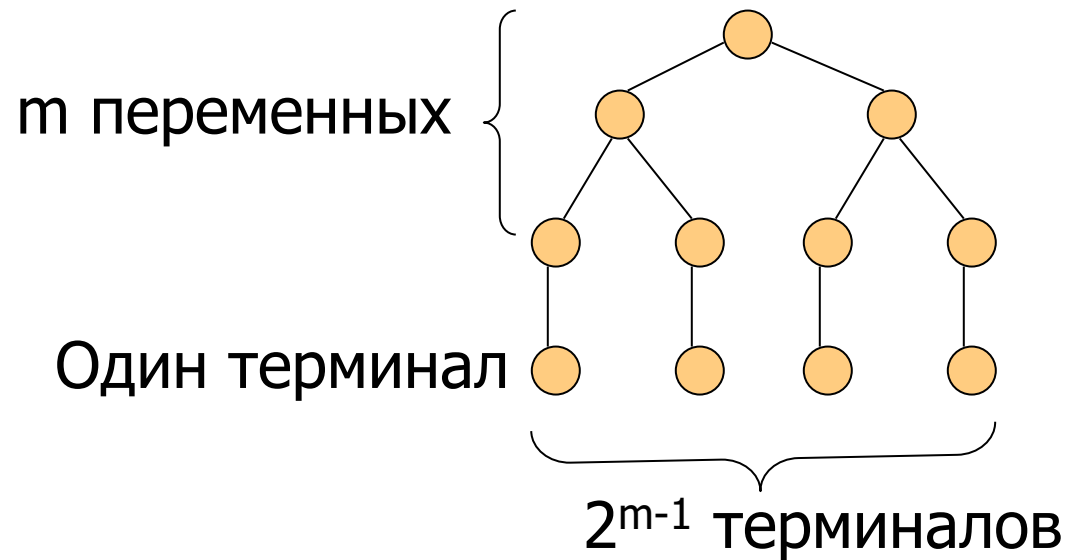
1.  $|vwx| \leq n$ .
2.  $|vx| > 0$ .
3. Для всех  $i \geq 0$ ,  $uv^iwx^i$  принадлежит  $L$ .

## Доказательство леммы о накачке

- Начнём с грамматики в НФХ для  $L - \{\epsilon\}$ .
- Пусть грамматика имеет  $m$  переменных (нетерминалов).
- Возьмём  $n = 2^m$ .
- Пусть  $z$  длины  $\geq n$ , будет в  $L$ .
- Докажем, что (“*лемма*”) дерево разбора с кроной  $z$  должно иметь путь длины  $m+2$  или более от корня к листьям кроны.

# Доказательство леммы 1

- Если все пути в дереве разбора грамматики в НФХ имеют длину  $\leq t+1$ , то наибольшей кроной будет строка длины  $2^{t-1}$ , как в:

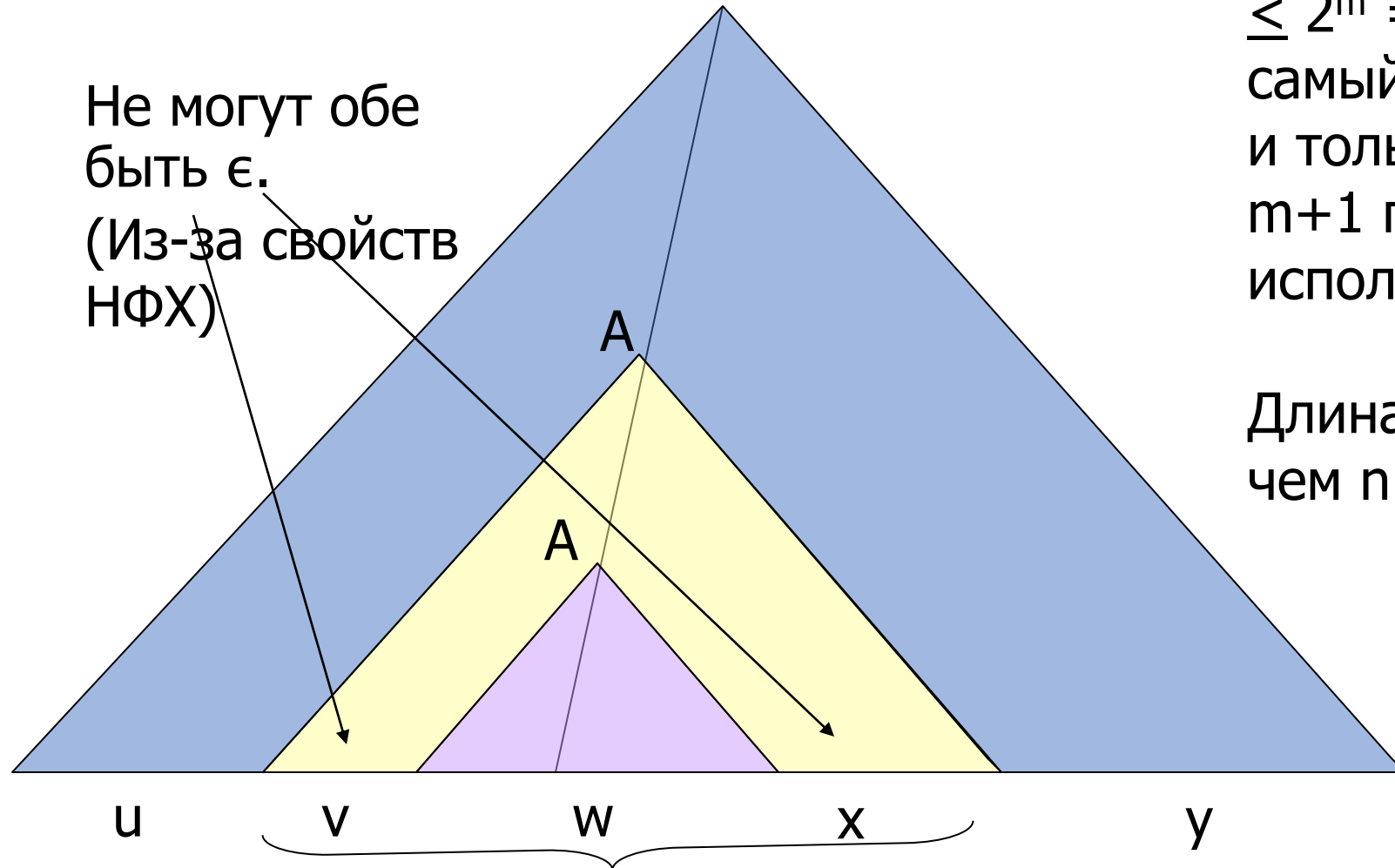




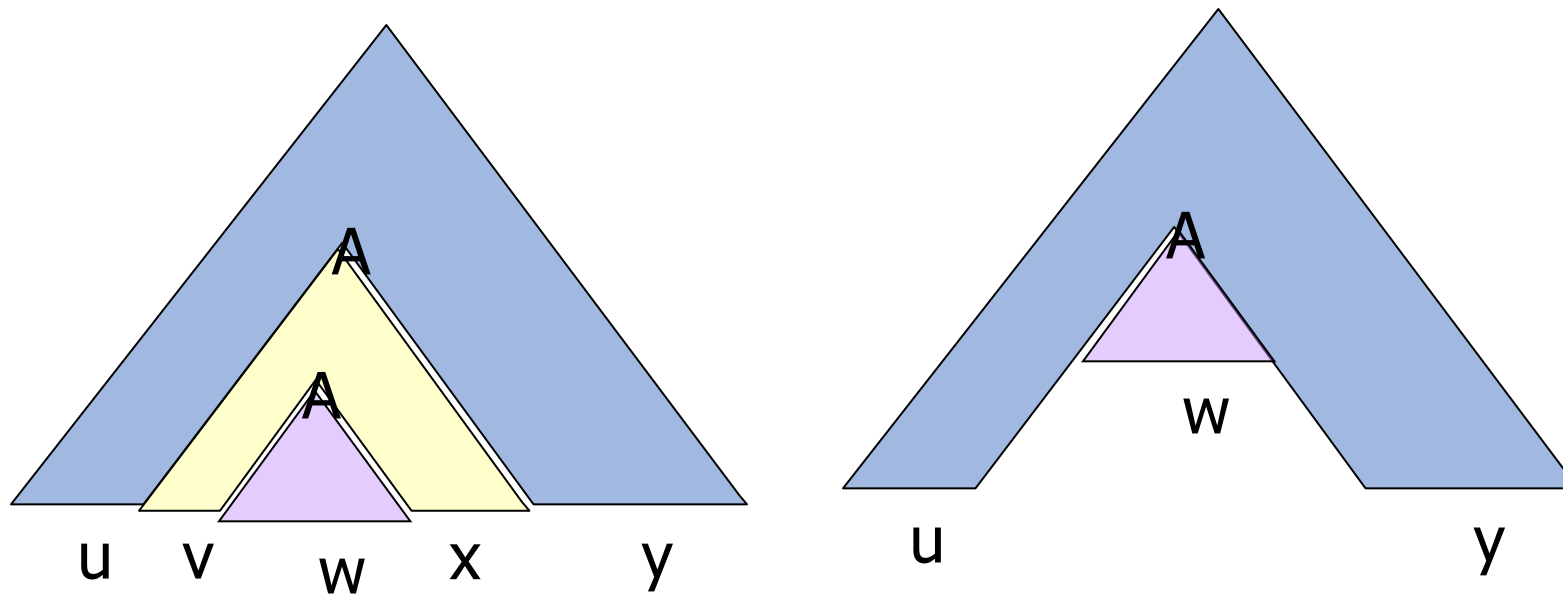
# Возврат к доказательству леммы о накачке

- Теперь мы знаем, что дерево разбора для  $z$  имеет путь, по крайней мере, из  $m+1$  переменных.
- Рассмотрим некоторый самый длинный путь.
- Есть только  $m$  различных переменных, так что среди  $m+1$  **на самом длинном пути** мы можем найти два узла с одной и той же меткой, скажем,  $A$ .
- Дерево разбора, таким образом, выглядит примерно так:

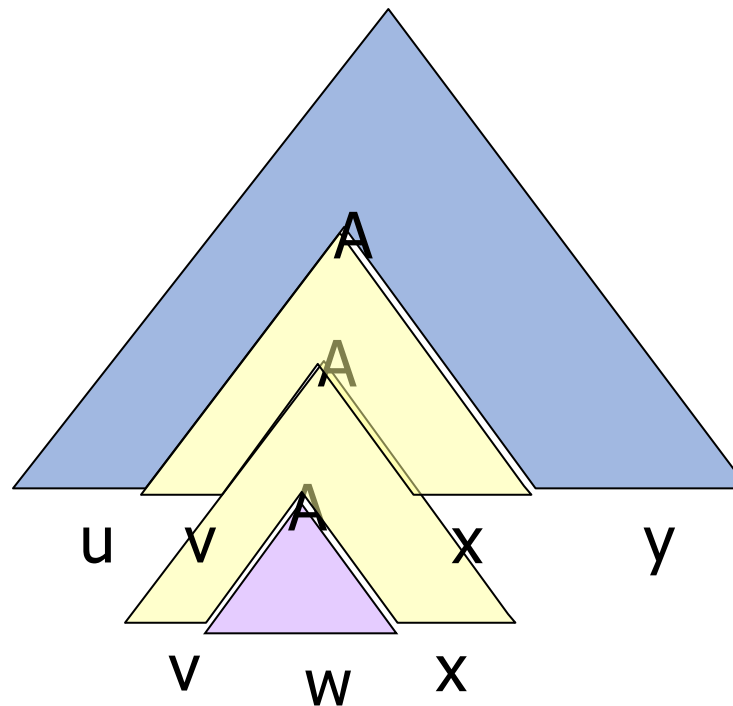
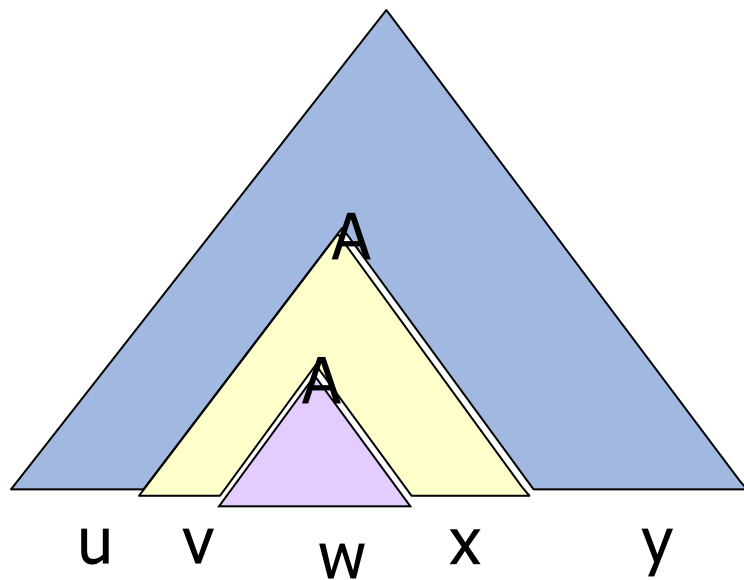
# Дерево разбора в доказательстве леммы о накачке



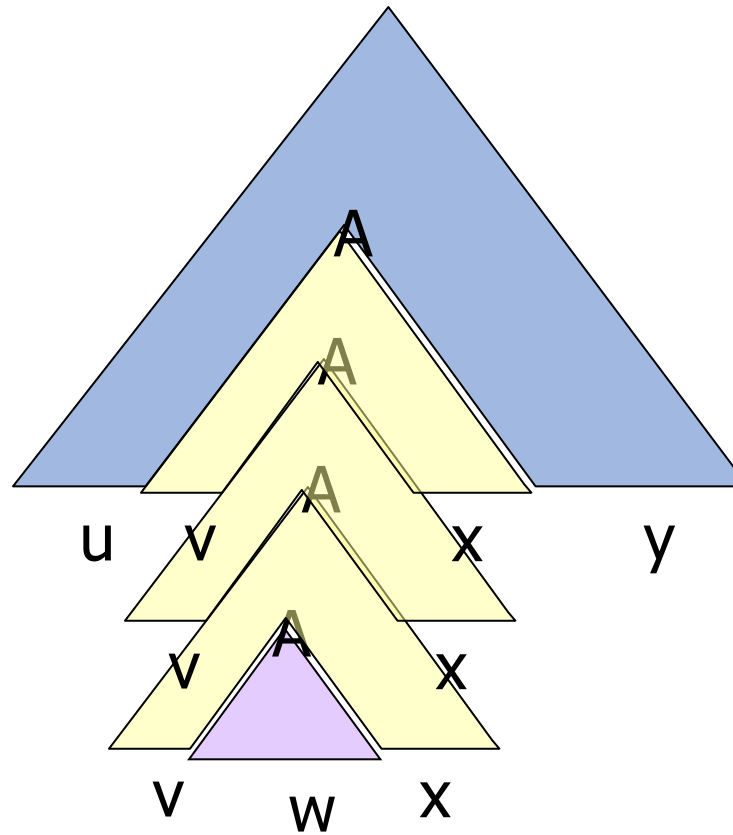
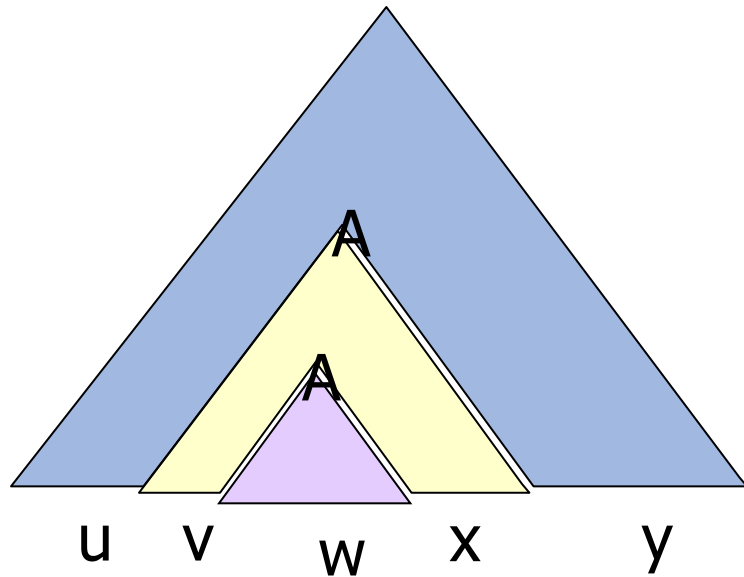
# Накачка нуль раз



# Накачка два раза



Накачка три раза и т.д.



# Использование леммы о накачке

- $\{0^i 10^i \mid i \geq 1\}$  есть КС-язык.
  - Мы можем отметить одну пару счетчиков.
- Но  $L = \{0^i 10^i 10^i \mid i \geq 1\}$  - нет.
  - Мы не можем выделить две пары, или три счетчика как группу.
- Доказывается с использованием леммы о накачке.
- Предположим, что  $L$  – КС-язык.
- Пусть  $n$  будет константой леммы о накачке для языка  $L$ .

## Использование леммы о накачке – (2)

- Рассмотрим  $z = 0^n 10^n 10^n$ .
- Мы можем записать  $z = uvwxu$ , где  $|vwx| \leq n$ , и  $|vx| \geq 1$ .
- **Случай 1:**  $vx$  не имеет 0-ей.
  - Тогда, по крайней мере, одна из них имеет 1, и  $uvw$  имеет, по крайней мере, одну 1, что при накачке не даёт строку из  $L$ .

## Использование леммы о накачке – (3)

- Мы все еще рассматриваем  $z = 0^n 10^n 10^n$ .
- **Случай 2:**  $vx$  имеет, по крайней мере, один 0.
  - $vwx$  слишком коротка (длина  $\leq n$ ), чтобы расширить все три блока 0-ей в  $0^n 10^n 10^n$ .
  - Рассмотрим строку  $uvw$ , которая, если L-КС-язык, должна быть в L.
  - НО,  $uvw$  имеет, по крайней мере, один блок из  $n$  0-ей, и, по крайней мере, один блок с меньшим, чем  $n$  0-ями.
  - Таким образом,  $uvw$  не входит в L.
  - Значит, наше предположение о контекстно-свободности языка L неверно!



# Свойства КС-языков

Разрешающие свойства

Свойства замыкания

# Обзор разрешимых свойств

- Обычно, когда мы говорим о КС-языках , мы имеем в виду их представление в виде КС-грамматики или МП-автомата, принимающих строки по переходу в конечное состояние или опустошением стека.
- Существуют алгоритмы решающие:
  1. Принадлежит ли строка  $w$  КС-языку  $L$ .
  2. Является ли КС-язык пустым.
  3. Является ли КС-язык бесконечным.

# Неразрешимые свойства

- Многие вопросы, разрешимые для регулярных языков, не могут быть разрешимы для КС-языков.
- **Пример:** Являются ли два КС-языка одним и тем же?
- **Пример:** Являются ли два КС-языка несвязными (пересечение пусто)?
  - Как бы вы это сделали для регулярных языков?
- Нужна теория машин Тьюринга и неразрешимых задач, чтобы доказать отсутствие соответствующих алгоритмов.

# Проверка пустоты

- Мы уже делали это.
- Мы научились устранять бесполезные переменные.
- Если стартовый символ есть среди них, то КС-язык пуст, в противном случае – нет.

# Проверка членства

- Необходимо узнать, принадлежит ли строка  $w$  языку  $L(G)$ .
- Предположим  $G$  находится в НФХ.
  - Или преобразуем данную грамматику в НФХ.
  - $w = \epsilon$  является специальным случаем, разрешаемым посредством проверки того, является ли начальный символ пустым.
- Алгоритм (СЮК) является хорошим примером динамического программирования и работает время порядка  $O(n^3)$ , где  $n = |w|$ .

СЮК = John Cocke, Dan Younger, and Tadao Kasami

# СУК алгоритм

- Пусть  $w = a_1 \dots a_n$ .
- Мы строим треугольный массив множеств переменных со стороной  $n$ .
- В матрице на месте  $(i, j)$   $X_{ij} = \{\text{переменные } A \mid A \Rightarrow^* a_i \dots a_j\}$ .
- Индукция по длине выводимой строки  $j-i+1$ .
  - Длина выводимой строки.
- Начинаем с  $X_{ii}$ , который выводит  $a_i$
- Затем находим  $X_{ii+1}$ , который выводит  $a_i a_{i+1}$
- И т.д.
- Наконец, находим  $X_{1n}$  и спрашиваем, есть ли  $S$  в  $X_{1n}$ .

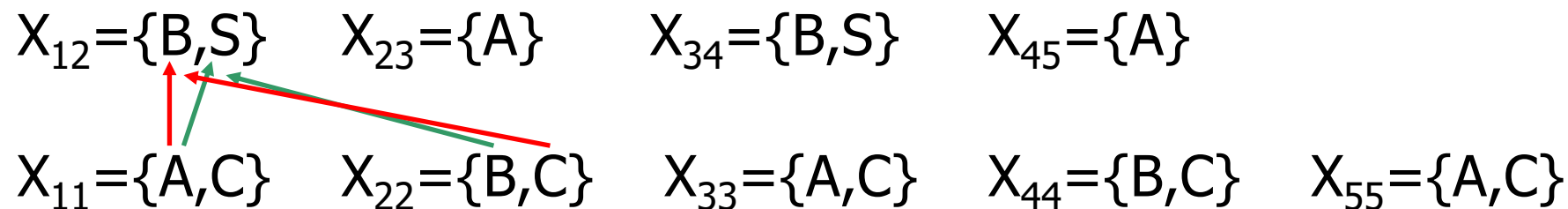
## СҮК алгоритм – (2)

- **Базис:**  $X_{ii} = \{A \mid A \rightarrow a_i \text{ есть продукция}\}$ .
- **Индукция:**  $X_{ij} = \{A \mid \text{существует продукция } A \rightarrow BC \text{ и целое } k, \text{ с } i \leq k < j, \text{ такое, что } B \text{ есть в } X_{ik} \text{ и } C \text{ есть в } X_{k+1,j}\}$ .

## Пример: CYK алгоритм

Грамматика:  $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$

Строка  $w = ababa$

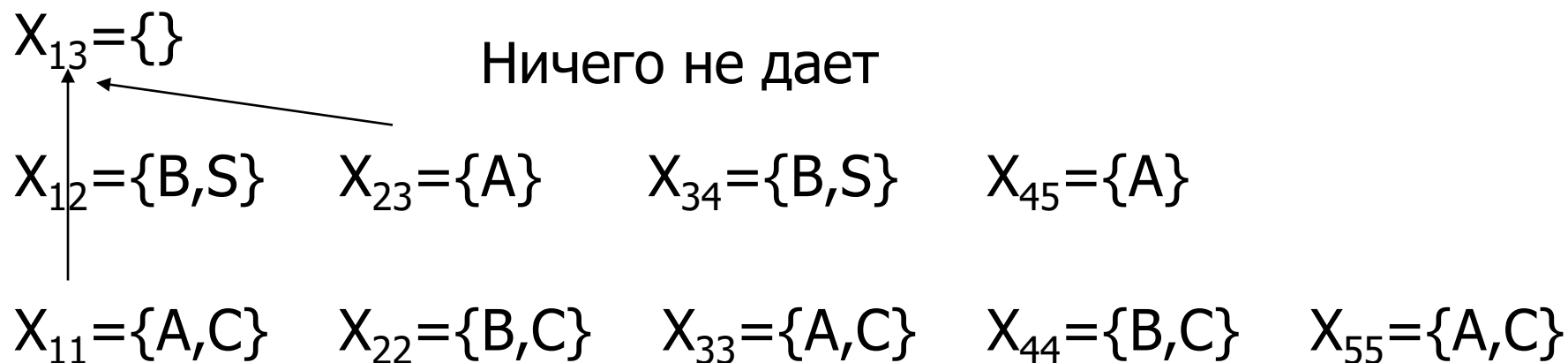




## Пример: CYK алгоритм

Грамматика:  $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$

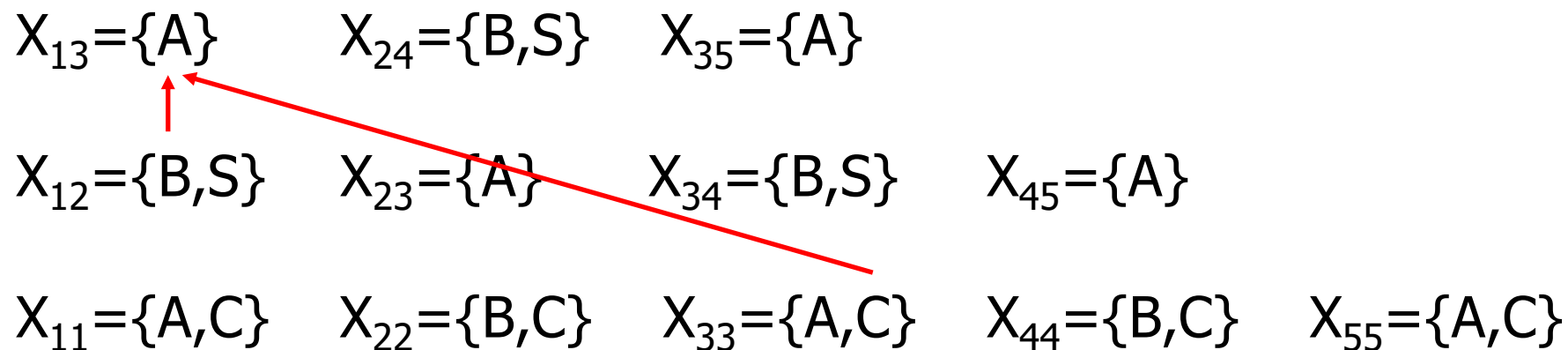
Строка  $w = ababa$



## Пример: CYK алгоритм

Грамматика:  $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$

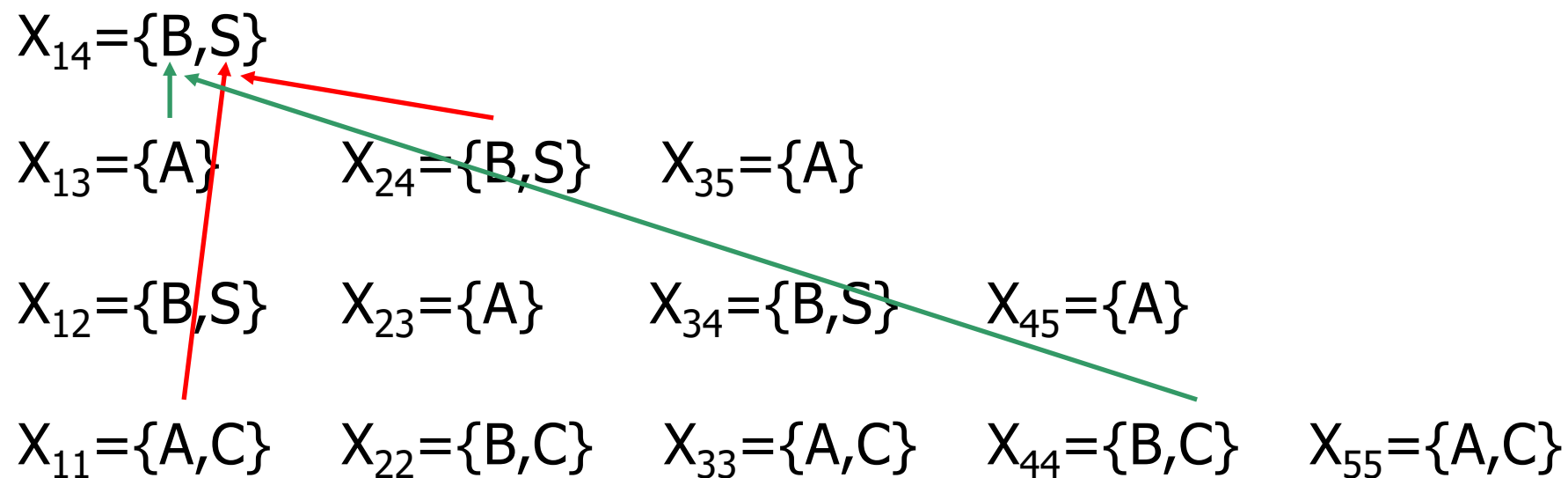
Строка  $w = ababa$



# Пример: CYK алгоритм

Грамматика:  $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$

Строка  $w = ababa$



# Пример: CYK алгоритм

Грамматика:  $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$

Строка  $w = ababa$

$X_{15} = \{A\}$

$X_{14} = \{B, S\}$

$X_{25} = \{A\}$

$X_{13} = \{A\}$

$X_{24} = \{B, S\}$

$X_{35} = \{A\}$

$X_{12} = \{B, S\}$

$X_{23} = \{A\}$

$X_{34} = \{B, S\}$

$X_{45} = \{A\}$

$X_{11} = \{A, C\}$

$X_{22} = \{B, C\}$

$X_{33} = \{A, C\}$

$X_{44} = \{B, C\}$

$X_{55} = \{A, C\}$

Т.к.  $S$  нет в  $X_{15}$ , мы заключаем, что строки  $ababa$  нет в языке данной грамматики.

# Проверка бесконечности

- Идея по сути та же, что и для регулярных языков.
- Использовать константу  $n$  леммы о накачке.
- Если существует строка языка длины между  $n$  и  $2n-1$ , то язык - бесконечный; в противном случае - нет.

# Свойства замыкания для КС-языков

- КС-языки замкнуты относительно операций объединения, конкатенации и замыкания Клини (т.е. относительно регулярных операций).
- Также, КС-языки замкнуты относительно обратимости строк, гомоморфизма и обратного гомоморфизма.
- Но не по пересечению или разности.

# Замыкание КС-языков по объединению

- Пусть  $L$  и  $M$  – КС-языки с грамматиками  $G$  и  $H$ , соответственно.
- Предположим  $G$  и  $H$  не имеют общих переменных.
  - Имена переменных не влияют на язык.
- Пусть  $S_1$  и  $S_2$  – начальные символы  $G$  и  $H$ .

## Замыкание КС-языков по объединению – (2)

- Образует новую грамматику для  $L \cup M$  путём комбинации всех символов и продукций  $G$  и  $H$ .
- Затем, добавим новый начальный символ  $S$ .
- Добавим продукции  $S \rightarrow S_1 \mid S_2$ .



## Замыкание КС-языков по объединению – (3)

- В построенной новой грамматике, все выводы начинаются с  $S$ .
- На первом шаге происходит замена  $S$  на  $S_1$  или  $S_2$ .
- В первом случае, результатом должна стать строка в  $L(G) = L$ , а во втором – строка в  $L(H) = M$ .

# Замыкание КС-языков по конкатенации

- Пусть  $L$  и  $M$  – КС-языки с грамматиками  $G$  и  $H$ , соответственно.
- Предположим  $G$  и  $H$  не имеют общих переменных .
- Пусть  $S_1$  и  $S_2$  – начальные символы  $G$  и  $H$ .

## Замыкание КС-языков по конкатенации– (2)

- Образует новую грамматику для  $LM$ , начав с того, что включим в неё все символы и productions  $G$  и  $H$ .
- Добавим новый начальный символ  $S$ .
- Добавим production  $S \rightarrow S_1S_2$ .
- Каждый вывод из  $S$  даст в результате строку в  $L$  с последующей за ней строкой из  $M$ .

# Замыкание КС-языков по операции Клини

- Пусть  $L$  имеет грамматику  $G$ , с начальным символом  $S_1$ .
- Образует новую грамматику для  $L^*$  путём введения в  $G$  нового начального символа  $S$  и продукций  $S \rightarrow S_1 S \mid \epsilon$ .
- Правый вывод из  $S$  генерирует последовательность из нуля или более переменных  $S_1$ , каждая из которых генерирует некоторую строку из  $L$ .

# Замыкание КС-языков по обратимости

- Если  $L$  – КС-язык с грамматикой  $G$ , образуем грамматику для  $L^R$  путём обращения тела каждой продукции.
- **Пример:**
  - Пусть  $G$  имеет  $S \rightarrow 0S1 \mid 01$ .
  - Обращение языка  $L(G)$  имеет грамматику  $S \rightarrow 1S0 \mid 10$ .

# Замыкание КС-языков по гомоморфизму

- Пусть  $L$  – КС-язык с грамматикой  $G$ .
- Пусть  $h$  - гомоморфизм на терминальных символах  $G$ .
- Построим грамматику для  $h(L)$ , заменяя каждый терминальный символ  $a$  на  $h(a)$ .

## Пример: Замыкание КС-языков по гомоморфизму

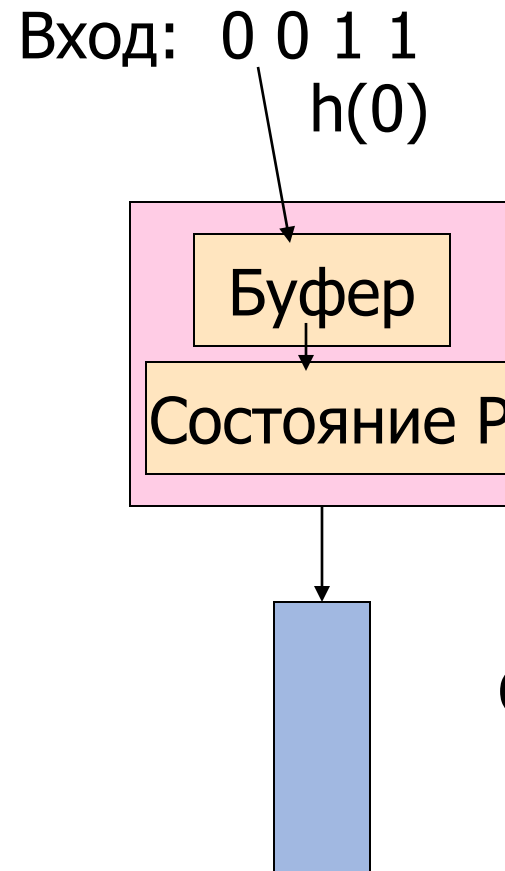
- $G$  имеет продукции  $S \rightarrow 0S1 \mid 01$ .
- $h$  определяется как  $h(0) = ab$ ,  $h(1) = \epsilon$ .
- $h(L(G))$  имеет грамматику с правилами  $S \rightarrow abS \mid ab$ .

# Замыкание КС-языков по обратному гомоморфизму

- Здесь нам не помогут грамматики, но вполне подойдет конструкция МПА.
- Пусть  $L = L(P)$  для некоторого МПА  $P$ .
- Построим МПА  $P'$  принимающий  $h^{-1}(L)$ .
- $P'$  моделирует  $P$ , но сохраняет, в качестве одной из частей двух-компонентного состояния, буфер, который хранит результат применения  $h$  к одному входному символу.



# Архитектура P'



Читаем сначала  
оставшийся символ  
в буфере, как если бы  
это было входом для P.

Стек P

# Формальное построение $P'$

- Состояниями являются пары  $[q, w]$ , где:
  1.  $q$  есть состояние  $P$ .
  2.  $w$  есть суффикс  $h(a)$  для некоторого символа  $a$ .
    - Таким образом, только конечное число возможных значений для  $w$ .
- Стековые символы  $P'$  – те же, что и у  $P$ .
- Начальным состоянием  $P'$  является  $[q_0, \epsilon]$ .

## Построение $P'$ – (2)

- Входные символы  $P'$  – символы, к которым применяется  $h$ .
- Финальные состояния  $P'$  – те состояния  $[q, \epsilon]$ , для которых  $q$  является финальным состоянием  $P$ .

# Переходы $P'$

1.  $\delta'([q, \epsilon], a, X) = \{([q, h(a)], X)\}$  для любого символа  $a$  автомата  $P'$  и любого стекового символа  $X$ .
  - Когда буфер пуст,  $P'$  может перезагрузить его.
2.  $\delta'([q, bw], \epsilon, X)$  содержит  $([p, w], \alpha)$ , если  $\delta(q, b, X)$  содержит  $(p, \alpha)$ , где  $b$  есть либо входной символ  $P$ , либо  $\epsilon$ .
  - Моделирование  $P$  из буфера.

# Доказательство корректности $P'$

- Нам нужно показать, что  $L(P') = h^{-1}(L(P))$ .
- **Ключевой аргумент:**  $P'$  делает переход  $([q_0, \epsilon], w, Z_0) \vdash^* ([q, x], \epsilon, \alpha)$  т.и.т.т., когда  $P$  делает переход  $(q_0, y, Z_0) \vdash^* (q, \epsilon, \alpha)$ ,  $h(w) = yx$ , и  $x$  есть суффикс последнего символа  $w$ .
- **Доказательство** в обоих направлениях проводится индукцией по числу произведенных шагов.

# Отсутствие замыкания по пересечению

- В отличие от регулярных языков, класс КС-языков не замкнут относительно  $\cap$ .
- Мы знаем, что  $L_1 = \{0^n 1^n 2^n \mid n \geq 1\}$  не есть КС-язык (использовали лемму о накачке).
- Однако,  $L_2 = \{0^n 1^n 2^i \mid n \geq 1, i \geq 1\}$  КС-языком является.
  - КСГ:  $S \rightarrow AB, A \rightarrow 0A1 \mid 01, B \rightarrow 2B \mid 2$ .
- Это так и для  $L_3 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$ .
- Но  $L_1 = L_2 \cap L_3$  !

# Отсутствие замыкания по разности

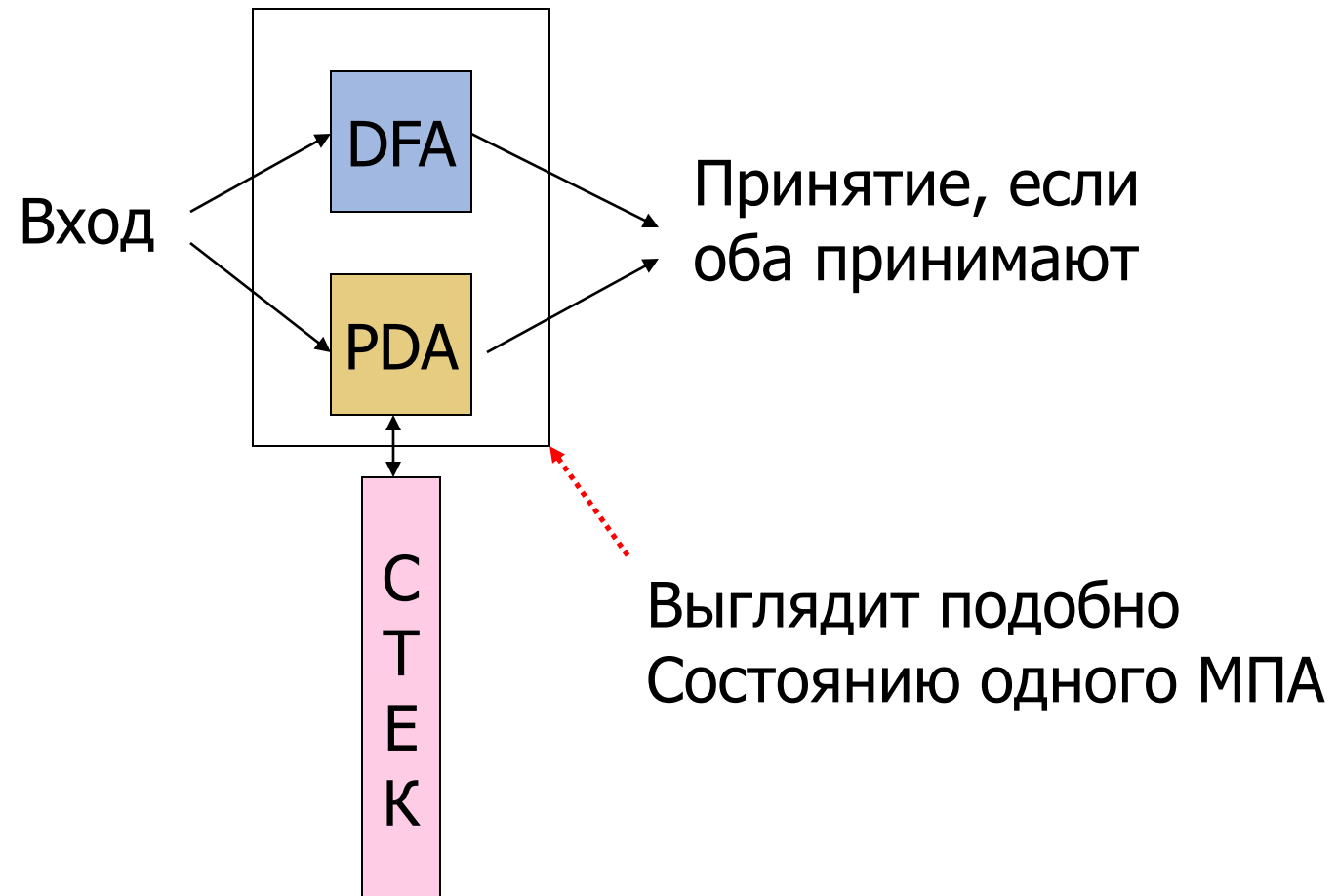
- Мы можем доказать нечто более общее :
  - Любой класс языков, замкнутый относительно разности, замкнут по пересечению.
- **Доказательство:**  $L \cap M = L - (L - M)$ .
- Таким образом, если бы КС-языки были замкнуты по разности, то они были бы замкнуты и по пересечению, но это не так.

# Пересечение с регулярными языками

- Пересечение двух КС-языков не обязательно будет КС-языком.
- Но пересечение КС-языка с регулярным языком всегда будет КС-языком.
- **Доказательство** включает запуск работы КДА в параллели с МПА, и отмечая, что их комбинация является МПА.
  - МПА принимает по переходу в заключительное состояние.



# КДА и МПА в параллели



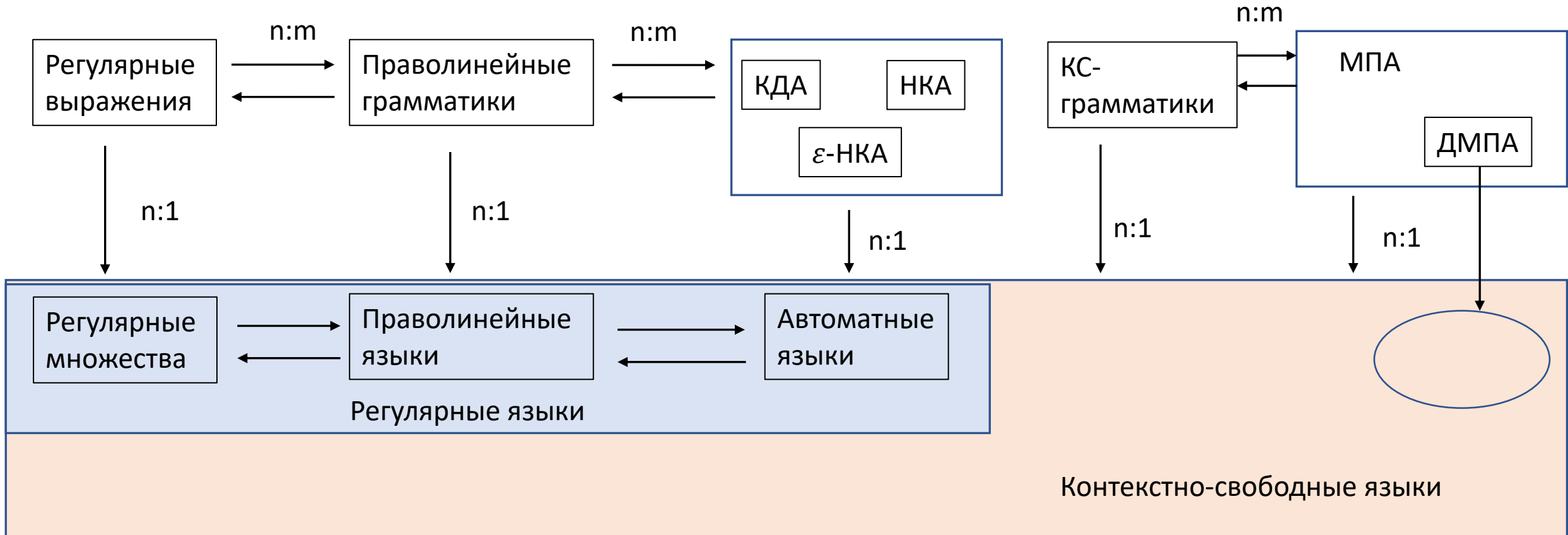
# Формальное построение

- Пусть КДА  $A$  имеет функцию переходов  $\delta_A$ .
- Пусть МПА  $P$  имеет функцию переходов  $\delta_P$ .
- Состояниями комбинированного МПА являются  $[q, p]$ , где  $q$  – состояние  $A$  и  $p$  - состояние  $P$ .
- $\delta([q, p], a, X)$  содержит  $([\delta_A(q, a), r], \alpha)$  если  $\delta_P(p, a, X)$  содержит  $(r, \alpha)$ .
  - Заметим, что в качестве  $a$  может быть  $\varepsilon$ , в этом случае  $\delta_A(q, a) = q$ .

## Формальное построение – (2)

- Финальными состояниями скомбинированного МП-автомата являются те  $[q, p]$ , в которых  $q$  есть финальное состояние  $A$  и  $p$  есть финальное состояние  $P$ .
- Начальным состоянием является пара  $[q_0, p_0]$ , состоящая из начальных состояний каждого автомата.
- **Простая индукция:**  $([q_0, p_0], w, Z_0) \vdash^* ([q, p], \varepsilon, \alpha)$  т.и.т.т., когда  $\delta_A(q_0, w) = q$  и в  $P$ :  $(p_0, w, Z_0) \vdash^* (p, \varepsilon, \alpha)$ .

# Подведём итоги



## Разрешимые свойства для регулярных языков

- Проблема принадлежности языку:  $w \in L$ ?
- Проблема пустоты:  $L = \emptyset$ ?
- Проблема эквивалентности:  $L = M$ ?
- Проблема вложения языков:  $L \subseteq M$ ?
- Проблема бесконечности языка:  $|L| = \infty$ ?

## Разрешимые свойства для КС-языков

- Принадлежит ли строка  $w$  КС-языку  $L$ .
- Является ли КС-язык пустым.
- Является ли КС-язык бесконечным.

## Неразрешимые свойства

- Проблема эквивалентности:  $L = M$ ?
- Проблема связности:  $L \cap M = \emptyset$ ?

## Свойства замкнутости для регулярных языков

- Регулярные языки замкнуты относительно операций объединения, конкатенации и замыкания Клини (регулярных операций)
- Замкнутость по обращению (слов)
- Замкнутость по дополнению
- Замкнутость по пересечению
- Замкнутость по вычитанию (разности)
- Замкнутость по гомоморфизму и обратному гомоморфизму

## Свойства замыкания для КС-языков

- КС-языки замкнуты относительно операций объединения, конкатенации и замыкания Клини (т.е. относительно регулярных операций).
- КС-языки замкнуты относительно обратимости строк,
- **КС-языки не замкнуты по пересечению**
- **КС-языки не замкнуты по разности.**
- КС-языки замкнуты относительно гомоморфизма и обратного гомоморфизма.
- Пересечение КС-языка с регулярным языком всегда будет КС-языком.

# Неразрешимость

Все есть целые числа

Счетные и несчетные множества

Машины Тьюринга

Рекурсивные и рекурсивно-перечислимые языки

# Целые числа, строки, и пр.

- Типы данных должны стать очень важными в качестве инструмента программирования.
- Но на другом уровне, есть только один тип, который можно понимать как целый или строковый..
- **Ключевая идея:** Строки, который представляют программы, только другой способ думать об одном и том же типе данных..



## Пример: Текст

- О строках ASCII или символах Unicode можно думать как бинарных строках 8 или 16 bits/символов.
- О бинарных строках можно думать как целых числах.
- Имеет смысл говорить о “ $i$ -ой строке” для  $i$ .

# Бинарные строки в целые числа

- Есть небольшой сбой :
  - Если думать просто о двоичных числах, то строки 101, 0101, 00101,... все окажутся «пятой строкой».
- Исправьте это путем добавления «1» к строке перед преобразованием в целое число.
  - Таким образом, 101, 0101 и 00101 станут 13-й, 21-й и 37-й строками соответственно.

## Пример: Изображения

- Представим изображение в виде (например) GIF.
- GIF-файл есть ASCII-строка.
- Конвертируем строку в бинарную строку.
- Конвертируем бинарную строку в целое число.
- Теперь у нас есть понятие «*i*-й образ».

## Пример: Доказательства

- Формальное доказательство - последовательность логических выражений, каждое из которых следует из предшествующих.
- Закодируем математические выражения любого типа в Unicode.
- Преобразуем выражение в двоичную строку, а затем в целое число.

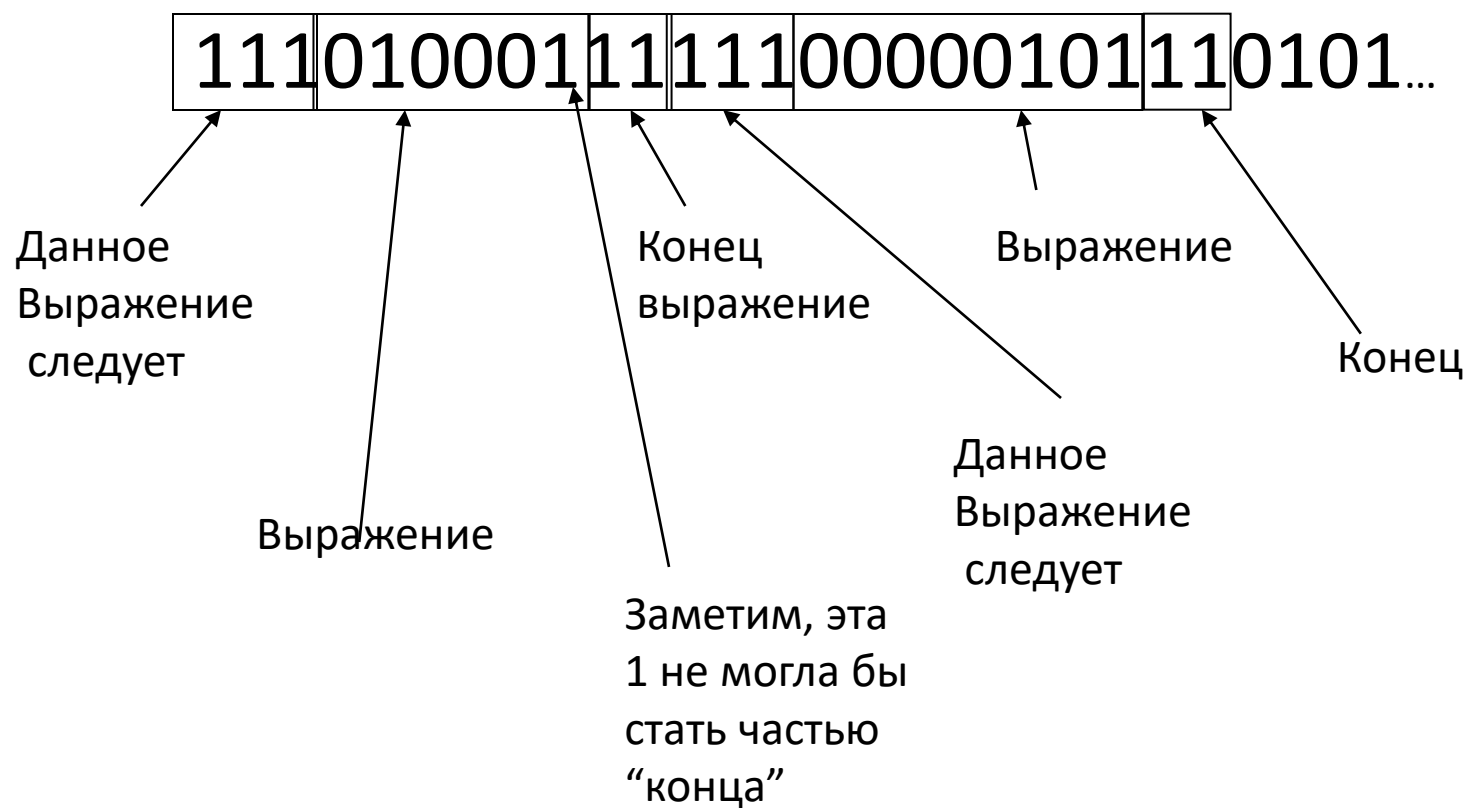
## Доказательства – (2)

- Но доказательство есть последовательность выражений, так что нам нужен способ разделить их.
- Также, нам нужно указать, какие выражения даны, а какие следуют из предыдущих.

# Доказательства – (3)

- Простой способ ввести новые символы в бинарные строки:
  1. Для данной бинарной строки, предварим каждый бит нулём 0.
    - Пример: 101 станет 010001.
  2. Используем строки из двух или более 1'ц как специальные символы.
    - Пример: 111 = “следующие выражения даны”; 11 = “конец выражения.”

# Пример: Закодированные доказательства



# Пример: Программы

- Программы - это просто другой вид данных.
- Представление программы в ASCII.
- Преобразуем в двоичную строку, затем в целое число.
- Таким образом, имеет смысл говорить о «i-й программе».
- Не так уж много программ.



# Конечные множества

- *Конечные множества* имеют особые числа, которые являются числом элементов.
- *Пример*:  $\{a, b, c\}$  – конечное множество; его *кардинал* есть 3.
- Конечное множество: Невозможно найти 1-1 отображение между элементами конечного множества и соответствующим подмножеством самого множества.

# Бесконечные множества

- Формально, *бесконечное множество* есть множество, для которого существует 1-1 соответствие между им самим и соответствующим подмножеством самого себя.
- **Пример:** положительные целые числа  $\{1, 2, 3, \dots\}$  есть бесконечное множество.
  - Существует 1-1 соответствие  $1 \leftrightarrow 2, 2 \leftrightarrow 4, 3 \leftrightarrow 6, \dots$  между этим множеством и соответствующим подмножеством (множеством четных чисел).

# Счётные множества

- *Счётные множества* есть множества с 1-1 соответствие с целыми положительными числами.
  - Следовательно, все счётные множества – бесконечные.
- **Пример:** Все целые.
  - $0 \leftrightarrow 1; -i \leftrightarrow 2i; +i \leftrightarrow 2i+1$ .
  - Таким образом, порядок будет следующим: 0, -1, 1, -2, 2, -3, 3,...
- **Примеры:** множество бинарных строк, множество Java-программ.

## Пример: Пары целых

- Упорядочим пары положительных чисел сначала по сумме, затем по первой компоненте:
- $[1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2], \dots, [1,4], [5,1], \dots$
- **Интересное исключение:** опишем функцию  $f(i,j)$  такую, что пара  $[i,j]$  соответствует целому  $f(i,j)$  в этом порядке

# Перечисления

- *Перечисление* множества - это 1-1 соответствие между the множеством и целыми положительными числами.
- Таким образом, мы уже определили перечисления для строк, программ, доказательств и пар целых чисел.

# Насколько много языков?

- Является ли множество языков над  $\{0,1\}$  счётным?
- Нет; приведем [доказательство](#).
- [Доказательство](#).
- Предположим мы смогли бы перечислить все языки над  $\{0,1\}$  и говорить о них как об “ $i$ -ом языке.”
- Рассмотрим язык  $L = \{ w \mid w - i\text{-ая бинарная строка и } w \text{ не есть } i\text{-ый язык} \}$ .

# Доказательство – прод.

- Ясно, что  $L$  есть язык над  $\{0,1\}$ .
- Таким образом, т.к. языки над  $\{0,1\}$  перечислимы, то это есть  $j$ -ый язык для некоторого  $j$ .
- Пусть  $x$  будет  $j$ -ой строкой.
- Находится ли  $x$  в  $L$ ?
  - Если да, то  $x \notin L$  по определению  $L$ .
  - Если нет, то  $x \in L$  по определению  $L$ .



# Доказательство — окончание

- Имеем противоречие:  $x$  принадлежит  $L$  и не принадлежит  $L$ , Таким образом, наше предположение (что существует перечисление языков) неверно.
- **Комментарий:** Это действительно плохо; языков больше, чем программ.
- Т.е., существуют языки без алгоритма определения членства.



# Рисунок диагонализации

		Строки					
		1	2	3	4	5	...
Языки	1	1	0	1	1	0	...
	2		1				
	3			0			
	4				0		
	5					1	
	...						...

# Рисунок диагонализации

Инвертируем  
каждый  
диагональный  
элемент

Языки

Строки

	1	2	3	4	5	...
1	0	0	1	1	0	...
2		0				
3			1			
4				1		
5					0	
...						...

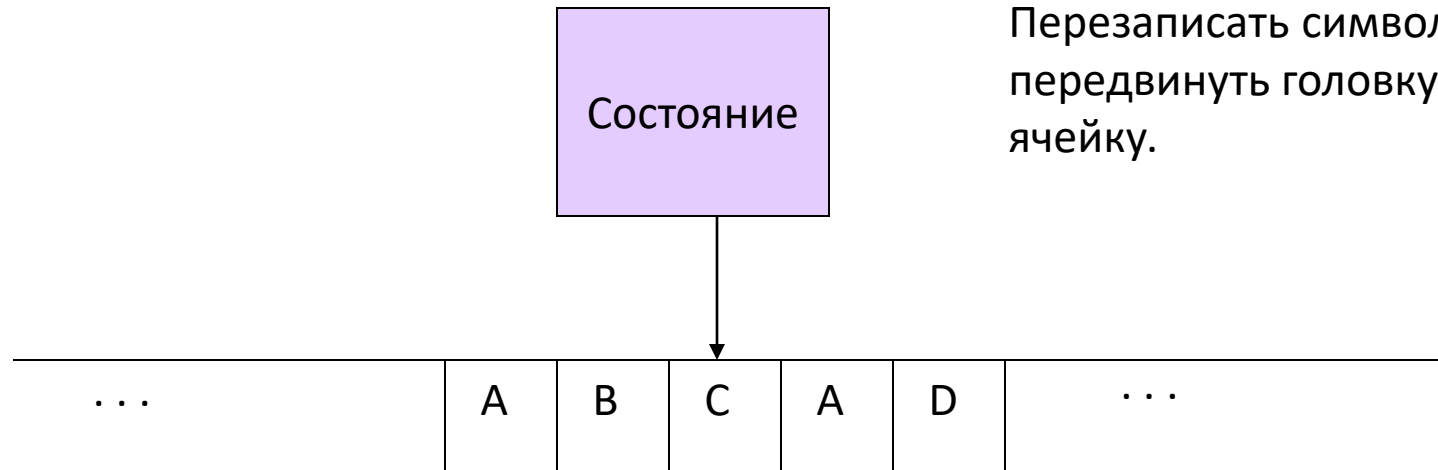
Не может быть  
строкой—  
Это не согласуется  
с элементами  
каждой строки.

# Теория машин Тьюринга

- Цель теории машин Тьюринга – доказать, что некоторые специфические языки не имеют алгоритмов.
- Начинают с языков для МТ.
- Неразрешимость доказывается путем сведения задачи к уже известной неразрешимой.

# Иллюстрация МТ

**Действие:** базируется на Состоянии и символе ленты, Обозреваемым читающей головкой: изменить состояние, Перезаписать символ и передвинуть головку на одну ячейку.



Бесконечная лента с ячейками, содержащими containing символы из некоторого конечного алфавита

# Формальное определение МТ

- ТМ определяется:
  1. Конечным множеством *состояний* ( $Q$ , обычно).
  2. *Входной алфавит* ( $\Sigma$ , обычно).
  3. *Алфавит ленты* ( $\Gamma$ , обычно; contains  $\Sigma$ ).
  4. *Функция переходов* ( $\delta$ , обычно).
  5. *Начальный символ* ( $q_0$ , in  $Q$ , обычно).
  6. *Пустой символ* ( $B$ , in  $\Gamma - \Sigma$ , обычно).
    - Вся лента, за исключением входа, первоначально пуста.
  7. Множество *конечных состояний* ( $F \subseteq Q$ , обычно).

# Функция переходов

- Имеет два аргумента:
  1. Состояние, в  $Q$ .
  2. Символ ленты в  $\Gamma$ .
- $\delta(q, Z)$  - либо не определена, либо является тройкой вида  $(p, Y, D)$ .
  - $p$  - состояние.
  - $Y$  – новый символ ленты.
  - $D$  - *направление*, L или R.

# Пример: Машина Тьюринга

- МТ: входы – символы 0 и 1.
- Данная МТ сканирует свой вход слева направо в поиске 1.
- Если она находит её, то изменяет на 0, переходит в финальное состояние  $f$ , и останавливается.
- Если она видит пробел, она изменяет его на 1 и сдвигается влево. Далее процесс повторяется вновь.

## Пример: Машина Тьюринга – (2)

- Состояния =  $\{q \text{ (start), } f \text{ (final)}\}$ .
- Входные символы =  $\{0, 1\}$ .
- Символы ленты =  $\{0, 1, B\}$ .
- $\delta(q, 0) = (q, 0, R)$ .
- $\delta(q, 1) = (f, 0, R)$ .
- $\delta(q, B) = (q, 1, L)$ .

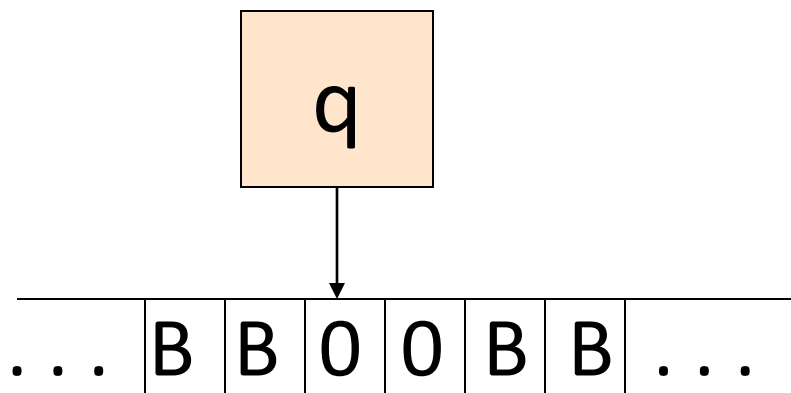


# Моделирование МТ

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

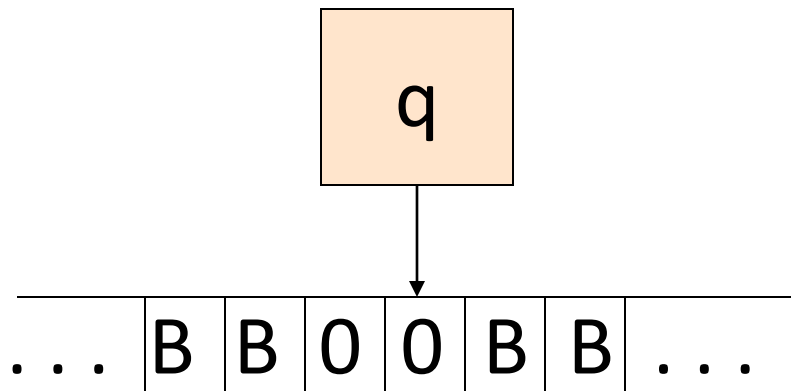


# Моделирование МТ

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

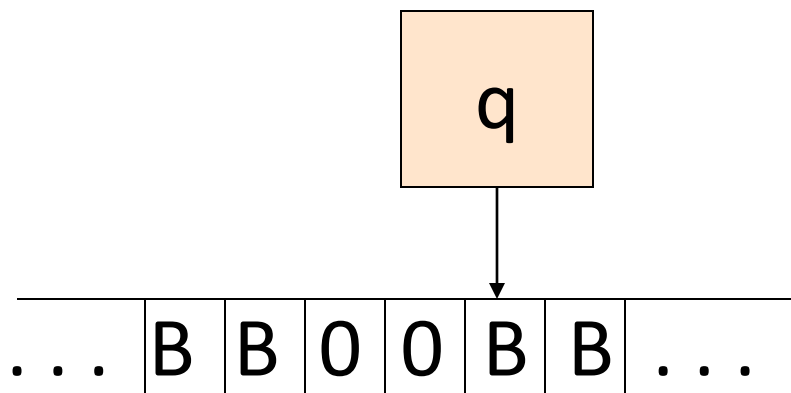


# Моделирование МТ

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

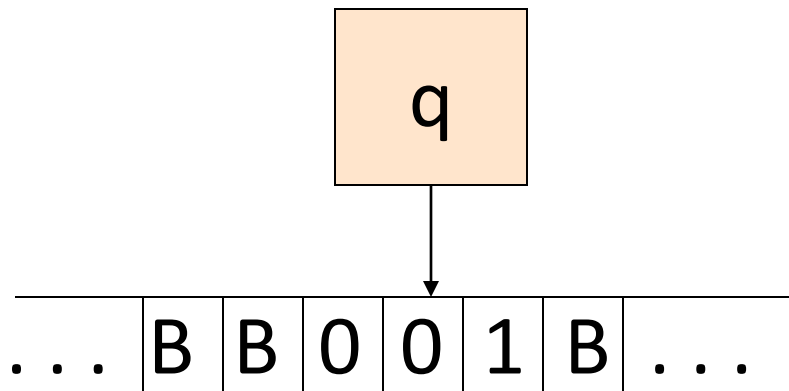


# Моделирование МТ

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

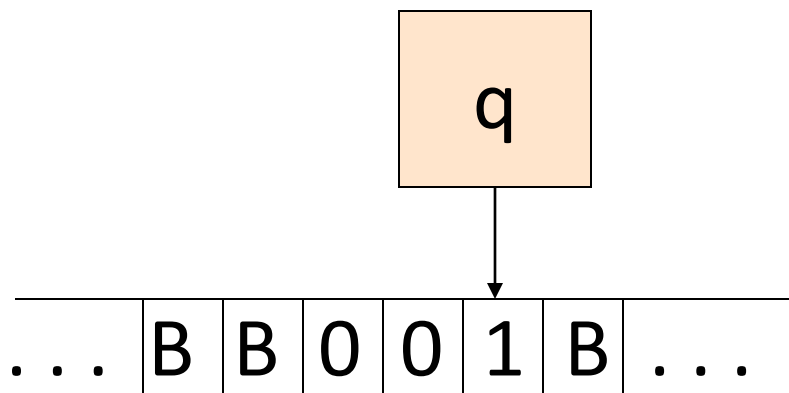


# Моделирование МТ

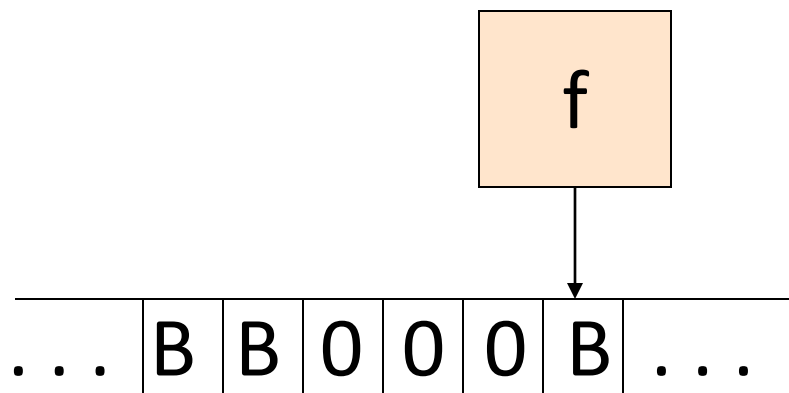
$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$



# Моделирование МТ



$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

Передвижения  
невозможны.  
МТ останавливается и  
принимает вход.

# Мгновенные конфигурации машины Тьюринга

- Первоначально, МТ имеет ленту, состоящую из строки входных символов, окруженных первоначально бесконечным числом пустых ячеек в обоих направлениях.
- МТ находится в начальном состоянии и головка обзревает самый левый входной символ.

## Мгновенные конфигурации МТ– (2)

- МК – это строка  $\alpha q \beta$ , где  $\alpha \beta$  содержимое ленты между самым левым и самым правым непробелами.
- Состояние  $q$  находится непосредственно слева от сканируемого головкой символа ленты.
- Если  $q$  находится с правого конца, то сканируется  $B$ .
  - Если  $q$  сканирует  $B$  слева, то следующие пробелы до конца справа от  $q$  являются частью  $\alpha$ .



# Мгновенные конфигурации МТ – (3)

- Как и для МП-автоматов мы можем использовать символы  $\vdash$  и  $\vdash^*$  для обозначения “смены конфигурации за один шаг” и “изменения конфигурации за ноль или более шагов,” соответственно.
- **Пример:** Изменения конфигураций предыдущей МТ можно описать так:

$q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$

# Формальное определение шага работы МТ

1. Если  $\delta(q, Z) = (p, Y, R)$ , то
  - $\alpha q Z \beta \vdash \alpha Y p \beta$
  - Если  $Z$  есть пробел  $B$ , то также  $\alpha q \vdash \alpha Y p$
2. Если  $\delta(q, Z) = (p, Y, L)$ , то
  - Для любого  $X$ ,  $\alpha X q Z \beta \vdash \alpha p X Y \beta$
  - Дополнительно,  $q Z \beta \vdash p V Y \beta$

# Языки МТ

- МТ обычно определяет язык по конечному состоянию.
- $L(M) = \{w \mid q_0 w \vdash^* I, \text{ где } I \text{ есть конфигурация с конечным состоянием}\}.$
- Или, МТ может определять язык по остановке.
- $N(M) = \{w \mid q_0 w \vdash^* I, \text{ нет возможных переходов из данной конфигурации } I\}.$

# Эквивалентность принятия и останова

1. Если  $L = L(M)$ , то существует МТ  $M'$  такая, что  $L = H(M')$ .
2. Если  $L = H(M)$ , то существует МТ  $M''$  такая, что  $L = L(M'')$ .

## Доказательство 1: Конечное состояние-> Останов

- Модифицируем  $M$ , чтобы получить  $M'$ , следующим образом:
  1. Для каждого конечного состояния  $M$ , удалим все переходы так, чтобы  $M$  останавливалась в этом состоянии.
  2. Избежим случайной остановки  $M'$ .
    - Введем новое состояние  $s$ , которое всегда приводит к сдвигу по ленте вправо; то есть  $\delta(s, X) = (s, X, R)$  для всех символов  $X$ .
    - Если  $q$  не является финальным состоянием, и  $\delta(q, X)$  не определено, то пусть  $\delta(q, X) = (s, X, R)$ .

## Доказательство 2: Останов $\rightarrow$ Финальное состояние

- Модифицируем  $M$ , чтобы получить  $M''$ , следующим образом :
  1. Введём новое состояние  $f$ , единственное финальное состояние для  $M''$ .
  2.  $f$  не имеет переходов.
  3. Если  $\delta(q, X)$  не определено для какого-либо состояния  $q$  и символа  $X$ , то определи его так:  $\delta(q, X) = (f, X, R)$ .

# Рекурсивно-перечислимые языки

- Классы языков, определяемые машинами Тьюринга с использованием конечных состояний и останова, являются эквивалентными.
- Этот класс языков называется *рекурсивно-перечислимыми языками*.
- Для данного класса языком МТ является полуразрешимым алгоритмом.

# Рекурсивные языки

- *Разрешимый алгоритм* - это МТ, принимающая вход индикацией по переходу в конечное состояние, и останавливающаяся на любом входе безотносительно того, принимается или нет вход.
- Если  $L = L(M)$  для некоторой МТ  $M$ , которая является алгоритмом, то говорят, что  $L$  – *рекурсивный язык*.
- Рекурсивные языки – это подкласс рекурсивно-перечислимых языков



## Пример: рекурсивные языки

- Каждый КС-язык – рекурсивный язык
- Почти все, что мы можем вообразить, - рекурсивный язык.

# Свойства замыкания для рекурсивных и рекурсивно-перечислимых языков

- Оба класса языков замкнуты относительно объединения, конкатенации, операции Клини, обращения, пересечения, обратного гомоморфизма.
- Класс рекурсивных языков замкнут по операциям вычитания и дополнения.
- Рекурсивно-перечислимые языки замкнуты по гомоморфизму.

# Не рекурсивно-перечислимые языки

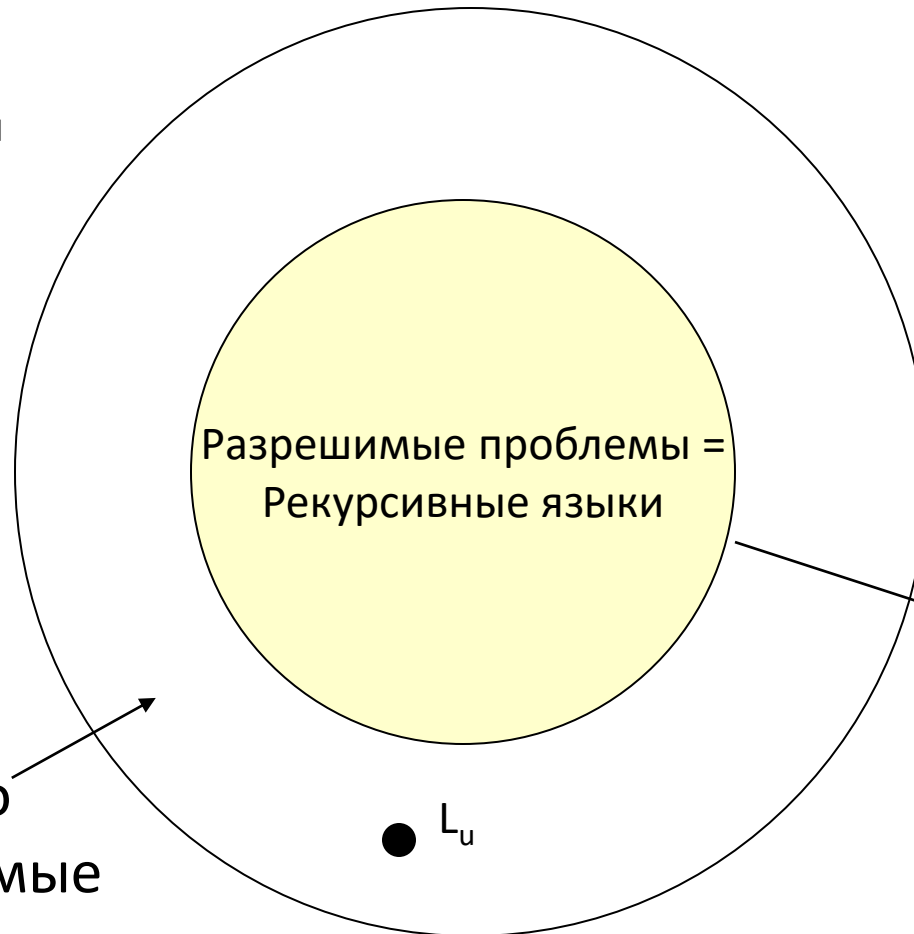
- Рассмотрим язык, состоящий из пар  $(M, w)$  таких, что:
  1.  $M$  – машина Тьюринга (закодированная бинарным кодом) с входным алфавитом  $\{0,1\}$ .
  2.  $w$  – строка из 0 и 1.
  3.  $M$  принимает вход  $w$ .
- Пусть  $L_d$  - бинарный язык, состоящий из всех строк  $v$  таких, что МТ  $M$ , код которой есть  $v$ , не принимает  $v$  в качестве своего входа.
- Не существует МТ, принимающей язык  $L_d$ .

# Универсальный язык

- Примером рекурсивно-перечислимого, но не рекурсивного языка является язык  $L_u$  *универсальной машины Тьюринга (УМТ)*.
- То есть, на вход УМТ подается код некоторой МТ  $M$  и некоторая бинарная строка  $w$ , и она принимает их, т.и.т.т., когда  $M$  принимает  $w$ .

Не рекурсивно-  
перечислимые  
языки

$L_d$



Разрешимые проблемы =  
Рекурсивные языки

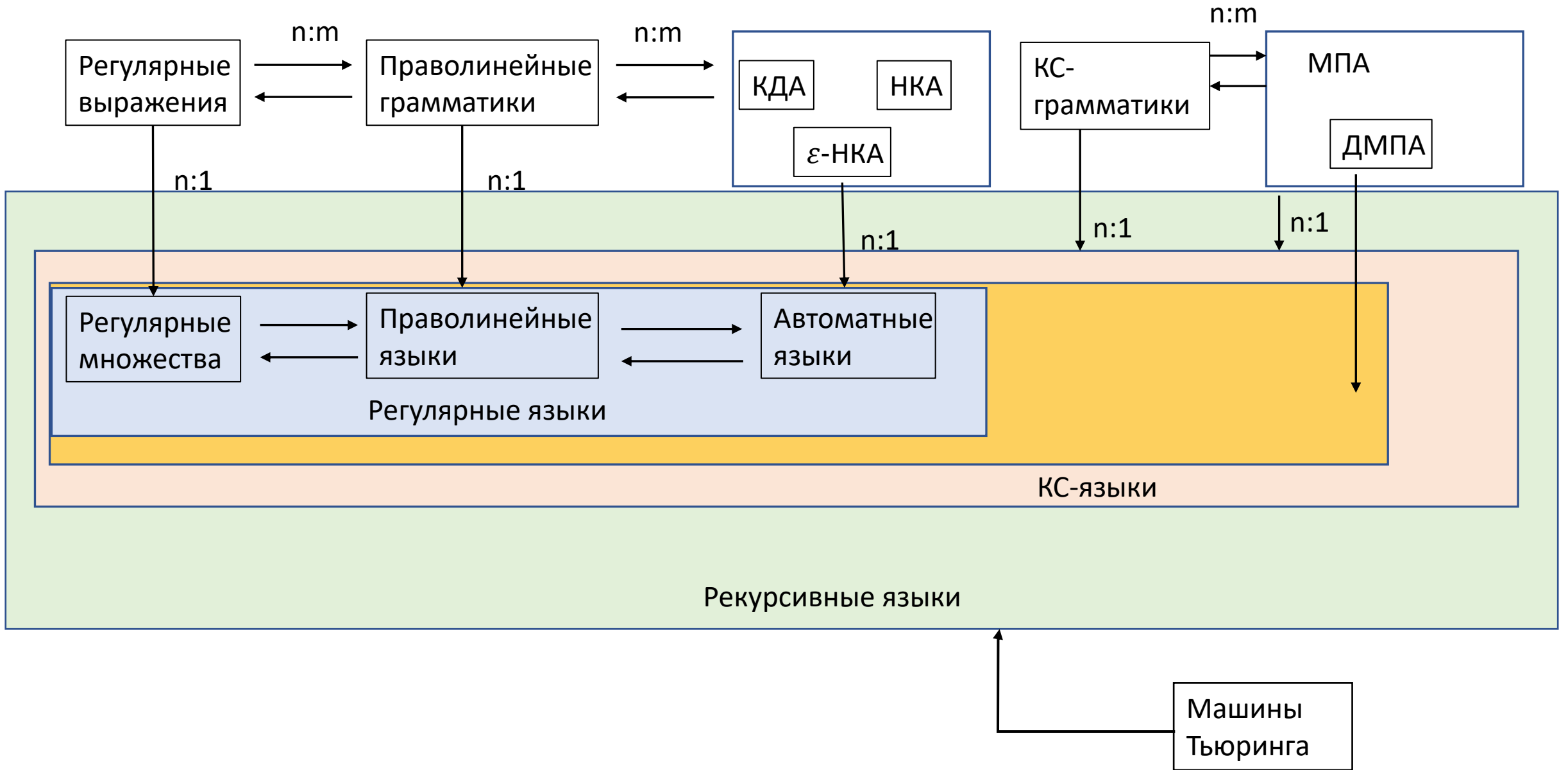
Все вне области –  
неразрешимые

Рекурсивно  
перечислимые  
языки

$L_u$

## Пример: Неразрешимые проблемы

- Можно ли выполнить определенную строку кода в программе?
- Неоднозначна ли данная контекстно-свободная грамматика?
- Генерируют ли две данных КС-грамматики один и тот же язык?
- Эквивалентен ли данный КС-язык языку  $\Sigma^*$ ?
- Является ли КС-язык регулярным?



Нормальная форма Грейбах



# Нормальная форма Грейбах

- Для каждого КС-языка можно найти грамматику, все правые части правил которых начинаются с терминалов.
- **Опр.** Нетерминал  $A$  грамматики  $G=(N,\Sigma,P,S)$  называется **рекурсивным**, если  $A \Rightarrow^* \alpha A \beta$  для некоторых  $\alpha$  и  $\beta$ . Если  $\alpha=\varepsilon$ , то  $A$  называется **леворекурсивным**, если  $\beta=\varepsilon$ , то **праворекурсивным**.
- Грамматика, имеющая хотя бы один леворекурсивный нетерминал, называется **леворекурсивной**.
- Аналогично определяется **праворекурсивная** грамматика.
- Грамматика, в которой все нетерминалы, кроме, может быть, начального символа, рекурсивные, называется **рекурсивной**.

# Устранение леворекурсивности

- Каждый КС-язык определяется хотя бы одной не леворекурсивной грамматикой.
- **Лемма.** Пусть  $G = (N, \Sigma, P, S)$  – КС-грамматика, в которой все  $A$ -правила имеют вид:

$$A \rightarrow A\alpha_1 | A\alpha_2 | \dots | A\alpha_m | \beta_1 | \beta_2 | \dots | \beta_n$$

и ни одна цепочка из  $\beta_i$  не начинается с  $A$ .

Пусть  $G' = (N \cup \{A'\}, \Sigma, P', S)$ , где  $A'$  - новый нетерминал, а  $P'$  получается из  $P$  заменой  $A$ -правил правилами:

$$\begin{aligned} A &\rightarrow \beta_1 | \beta_2 | \dots | \beta_n | \beta_1 A' | \beta_2 A' | \dots | \beta_n A' \\ A' &\rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_m | \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A'. \end{aligned}$$

Тогда  $L(G') = L(G)$ .

## Доказательство.

1.  $L(G') \subseteq L(G)$ . В  $G$  из  $A$  выводимы цепочки, определяемые РВ

$$(\beta_1 + \beta_2 + \dots + \beta_n)(\alpha_1 + \alpha_2 + \dots + \alpha_m)^*$$

Это в точности те цепочки, которые выводимы в  $G'$  из  $A$  с помощью правых выводов, применив один раз  $A$ -правило и несколько раз  $A'$ -правила.

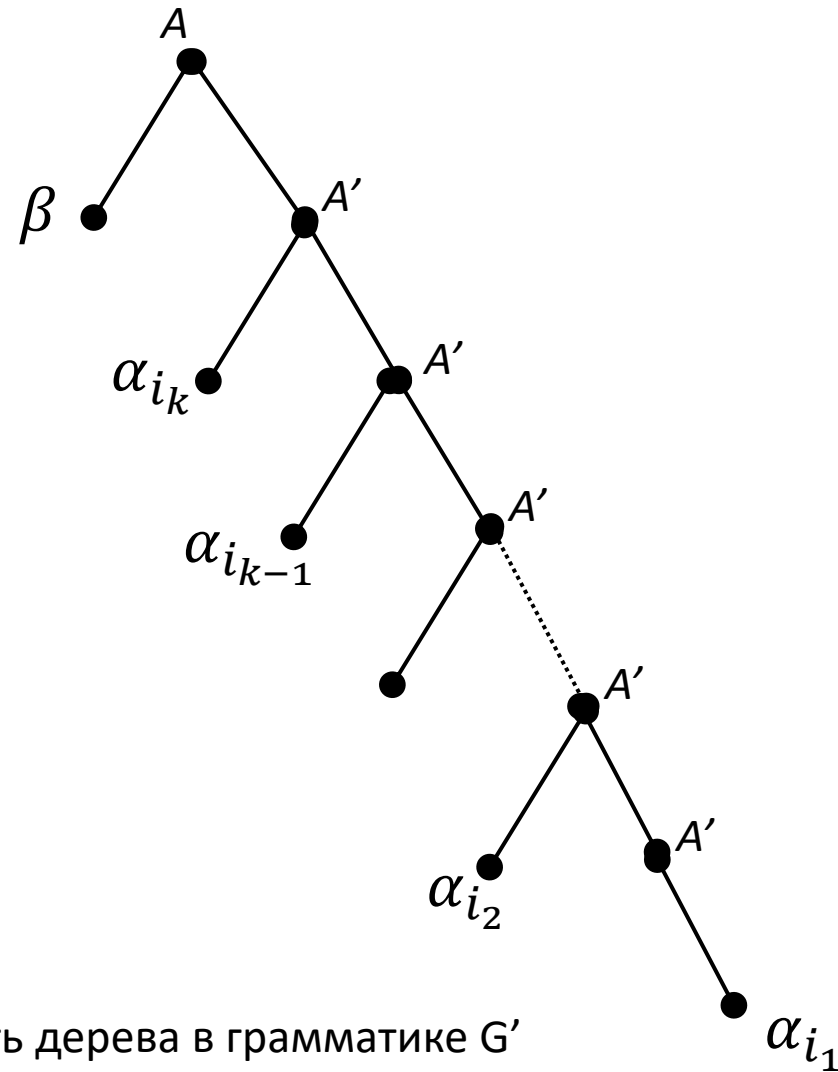
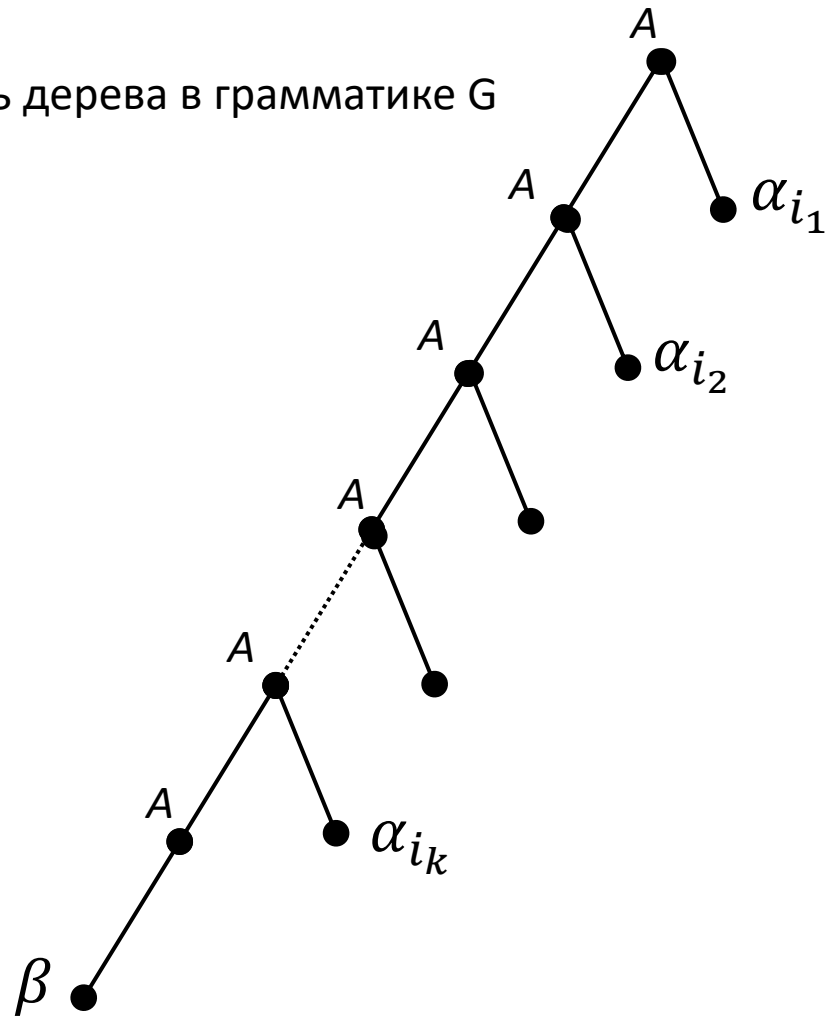
Все шаги вывода в  $G$ , в которых не используются  $A$ -правила, можно сделать и в  $G'$  (они одни и те же). Т.о.  $L(G') \subseteq L(G)$ .

2.  $L(G) \subseteq L(G')$  - показывается аналогично.

3. 
$$\begin{array}{l} L(G') \subseteq L(G) \\ L(G) \subseteq L(G') \end{array} \Rightarrow L(G') = L(G)$$

# Иллюстрация преобразования правил

Часть дерева в грамматике  $G$



Часть дерева в грамматике  $G'$

# Пример

- G:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid a$$

- G':

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

# Нормальная форма Грейбах

- **Теорема**. Каждый КС-язык определяется нелеворекурсивной грамматикой.
- **Доказательство** следует из эквивалентности преобразований правил.
- **Опр.** КС-грамматика  $G = (N, \Sigma, P, S)$  называется *грамматикой в нормальной форме Грейбах*, если в ней нет  $\varepsilon$ -правил и каждое правило из  $P$ , отличное от  $S \rightarrow \varepsilon$ , имеет вид  $A \rightarrow a\alpha$ , где  $a \in \Sigma$  и  $\alpha \in N^*$ .

- **Лемма**. Пусть  $G$  - не леворекурсивная грамматика. Существует такой порядок  $<$  на  $N$ , что если  $A \rightarrow B\alpha$  принадлежит  $P$ , то  $A < B$ .
- **Теорема**. Если  $L$ -КС-язык, то  $L = L(G)$  для некоторой грамматики  $G$  в нормальной форме Грейбах.