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), $T \cap N = \varnothing$;

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

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

α называется *левой частью (головой)* правила, β — *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

$$\alpha \to \beta_1 \quad \alpha \to \beta_2 \qquad \ldots \qquad \alpha \to \beta_n$$

используют сокращенную запись $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$.

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**

<u>Опр</u>. Грамматика $G = \langle T, N, P, S \rangle$ называется ***контекстно-свободной*** ($КС$), если каждое правило из P имеет вид $A \to \beta$, 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.
  - α, β, γ,… строки терминалов и/или переменных.

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

- Продукции:

  S -> 01

  S -> 0S1

- Базис: 01 есть в языке.
- Индукция: если w есть в языке, то есть 0w1.

Пример: Формальное задание КСГ

- Формальное определение КС-грамматики для $\{0^n1^n \mid n \geq 1\}$.
- Терминалы = {0, 1}.
- Переменные = {S}.
- Начальный символ = S.
- Продукции =

  S -> 01

  S -> 0S1

Примеры грамматик и языков

Контекстно-свободные

1. Грамматика
$$S \to aQb \mid accb$$
$$Q \to cSc$$

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

2. Грамматика
$$S \to aSa \mid bSb \mid \varepsilon$$

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):
$$S \to A \mid \varepsilon$$
$$A \to aAa \mid bAb \mid aa \mid bb$$

Выводы – интуитивно

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

Итеративный вывод

=>* означает “нуль или более шагов вывода.”

- Базис: α =>* α для любой строки α.
- Индукция: Если α =>* β и β => γ, то α =>* γ.

Пример: Итеративный вывод

- S -> 01; S -> 0S1.
- S => 0S1 => 00S11 => 000111.
- Таким образом,
- S =>* S; S =>* 0S1; S =>* 00S11; S =>* 000111.

Сентенциальные формы

- Любая строка переменных и/или терминалов, которая выводится из начального символа, называется а *сентенциальной формой*.
- Формально, α - есть сентенциальная форма, т.и.т.т., когда S =>* α.

Выводы – Формализм

- Мы говорим, что $\alpha A \beta \Rightarrow \alpha \gamma \beta$ если $A \to \gamma$ - есть продукция.
- Пример: S -> 01; S -> 0S1.
- S => 0S1 => 00S11 => 000111.

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 | S =>* w}.

- Пример: G имеет продукции S -> ϵ и S -> 0S1.
- $L(G) = \{0^n 1^n \mid n \geq 0\}$.

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 Джоном Бэкусом для описания языка Fortran и позднее АЛГОЛ в сообщении о языке АЛГОЛ 60
- Питер Наур был редактором сообщения, поэтому форму записи также называют формой Бэкуса - Наура.
- В БНФ слово обычно используется для описания переменной, например “statement”, если надо, чтобы эта переменная сгенерировала все строки, правильных предложений языка программирования
- Кроме того, терминалами являются не только отдельные символы, но и строки (например, ключевые слова).

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

- <число> ::= <чс>
- <чс> ::= <чс><цифра>|<цифра>
- <цифра> ::= 0|1|2|3|4|5|6|7|8|9

БНФ - нотация – (2)

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

БНФ – нотация – Замыкание Клини

- Символ … используется для “один или более.”
- Пример:

  <digit> ::= 0|1|2|3|4|5|6|7|8|9 <unsigned integer> ::= <digit>…

- Трансляция: Замена α… новой переменной A и продукциями A -> Aα | α.

Пример: Замыкание Клини

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

  U -> UD | D

  D -> 0|1|2|3|4|5|6|7|8|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 [α] новой переменной A с продукциями A -> α | ϵ.

Пример: Необязательные элементы

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

S -> iCtSA

A -> ;eS | ϵ

i = if

t = then

e = else

S – statement

C – 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 -> α.
- Использовать A вместо $\{\alpha\}$.

Пример: Группирование

L -> S [{;S}…]

- Заменяем на L -> S [A…]   A -> ;S
  - A обозначение для {;S}.
- Затем на L -> SB   B -> A… | ϵ   A -> ;S
  - B обозначение для [A…] (нуль или более A).
- Наконец, заменяем
- L -> SB   B -> C | ϵ   C -> AC | A   A -> ;S
  - C обозначение для A… .

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

`<U> ::= a{ab}*` - цепочка, начинающаяся с `a` и содержащая ноль или более (до бесконечности) *цепочек символов* `ab` ;

- `<U> ::= a{ab}+` - цепочка, начинающаяся с `a` и содержащая от одного до бесконечности повторений цепочки `ab` ;
- `<U> ::= 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: А-Яа-я - означает все литеры русского алфавита в кодировках ANSI cp 1251 и Unicode, а А-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

<число> ::= {<цифра>}+
<цифра> ::= 0-9
<врж> ::= [<врж>(\+|\-)]<терм>
<терм> ::= [<терм>(*|/)]<множ>
<множ> ::= [<множ>^]<степ>
<степ> ::= \(<врж>\)|<идентификатор>|<число>

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 -> β - продукция.
- Также, $\alpha \Rightarrow^*_{lm} \beta$ если α становится β в результате последовательности 0 или более $\Rightarrow_{lm}$ шагов.

- lm = leftmost

Пример: Левые выводы

- Грамматика сбалансированных скобок:

  S -> SS | (S) | ()

$S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$

- Таким образом, $S \Rightarrow^*_{lm} (())()$
- S => SS => S() => (S)() => (())() 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 α становится β в результате последовательности 0 или более $\Rightarrow_{rm}$ шагов.

- rm = rightmost

Пример: Правые выводы

- Грамматика сбалансированных скобок :

  S -> SS | (S) | ()

$S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$

- Таким образом, $S \Rightarrow^*_{rm} (())()$
- Но!

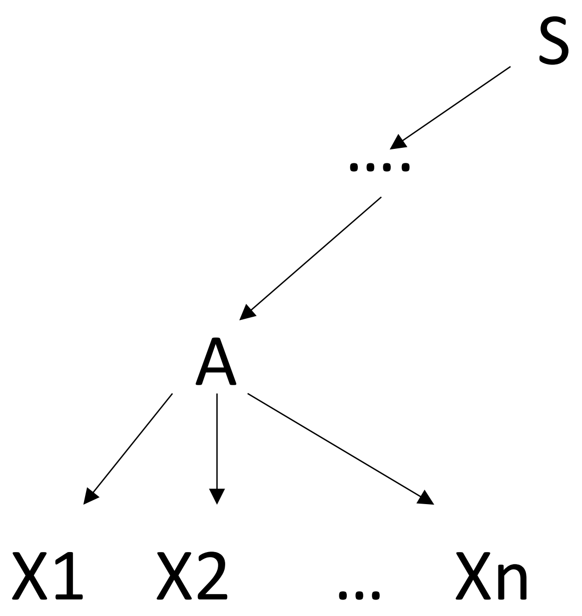
S => SS => SSS => S()S => ()()S => ()()() не левый и не правый вывод.

Деревья синтаксического разбора

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

Деревья выводов

- G = (N, Σ, P, S)
- A$\rightarrow$ X1, X2,…, Xn
- ……
-



Дерево вывода

- **Опр.** Помеченное упорядоченное дерево *D* называется *деревом вывода* (или *деревом разбора*) в КС-грамматике G = (N, Σ, 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 предками, то множество правил *P* допускает выводы $X_i \Rightarrow D_i$

   При этом поддерево D_i может быть узлом с меткой корня $Y_i = \varepsilon$ только в случае, если Y_i - единственный потомок узла *A* и в *P* присутствует правило $A \rightarrow \varepsilon$.

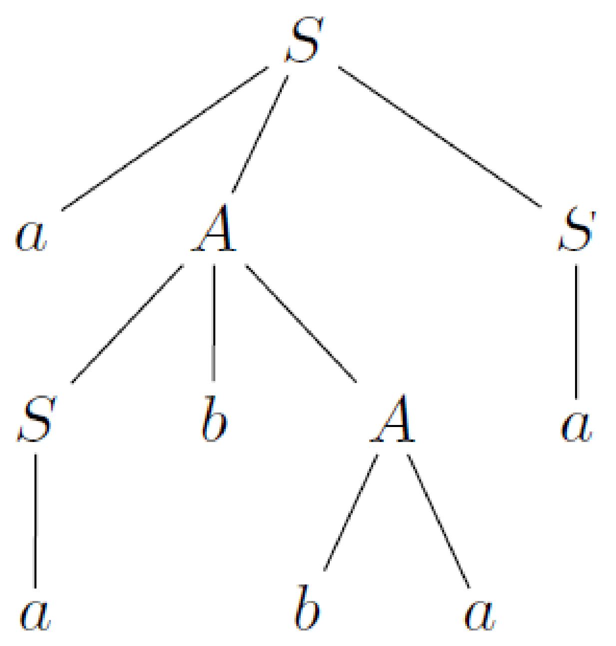
Пример дерева разбора

Рассм. G = ({S,A}, {a,b}, P, S),

где

P: $S \to aAS \mid a$

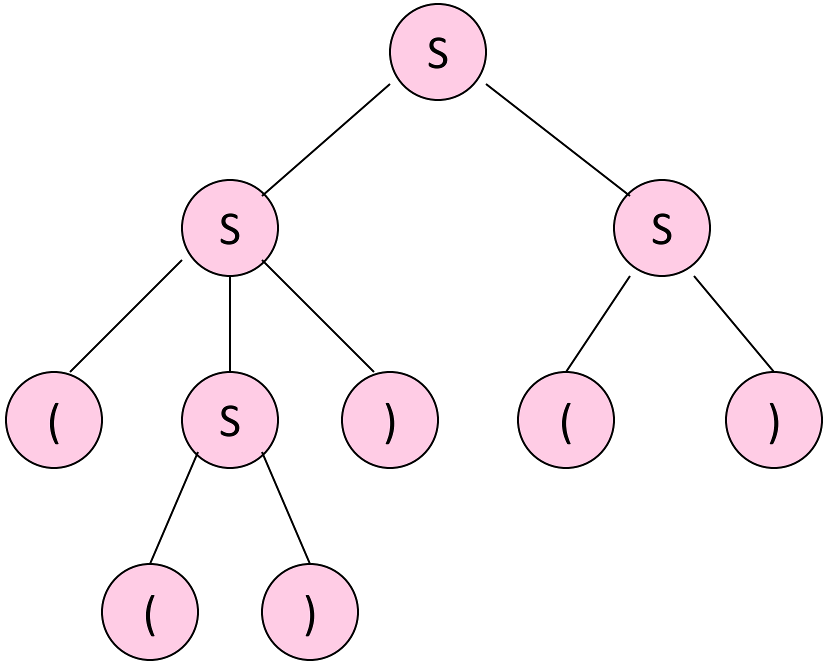
$\quad A \to SbA \mid ba \mid SS$



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

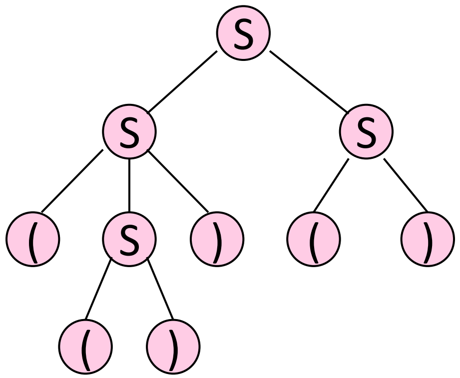
Пример: Дерево разбора

S -> SS | (S) | ()



Результат дерева разбора

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

- Пример: Крона  есть (())()

Сечение, крона сечения

- **<u>Опр</u>**. *Сечением* дерева D называется такое множество C вершин дерева D, что

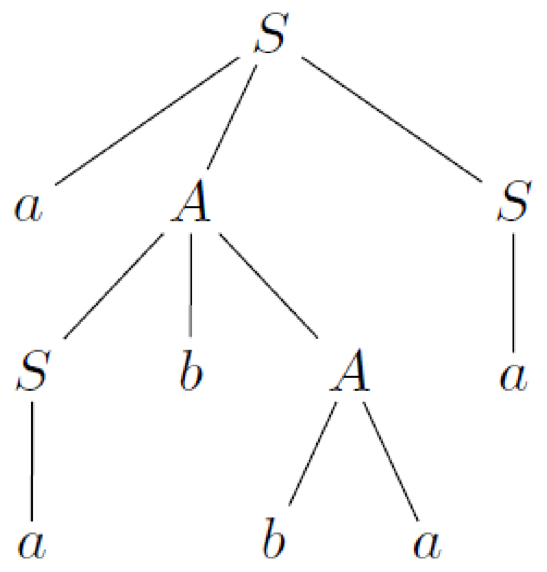
(1) Никакие две вершины из C не лежат на одном пути в D,

(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 ε, внутренний узел и его сыновья образуют правило вывода грамматики).
- Будем называть их *деревьями разбора с корнем A*.

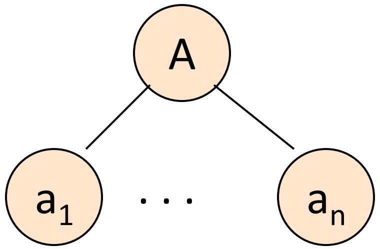
Деревья разбора, левый и правый выводы

- **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

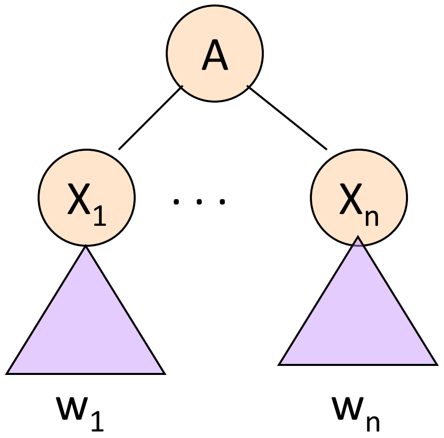
- Индукция по *высоте дерева* (длине наиболее длинного пути от корня).
- Базис: высота 1. Дерево выглядит так:



- $A \to a_1 \ldots a_n$ должно быть продукцией.
- Таким образом, $A \Rightarrow^*_{lm} a_1 \ldots a_n$.

Часть 1 – Индукция

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

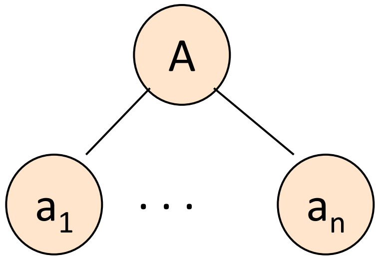


Доказательство: Часть 2

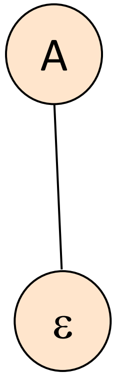
- По данному левому выводу терминальной строки нам нужно доказать существование дерева разбора.
- Доказательство проводится индукцией по длине вывода.

Часть 2 – Базис

- Если A =>$_{lm}$ $a_1 \ldots a_n$ вывод за один шаг, то должно существовать дерево разбора:



- Если A =>$_{lm}$ ε, то

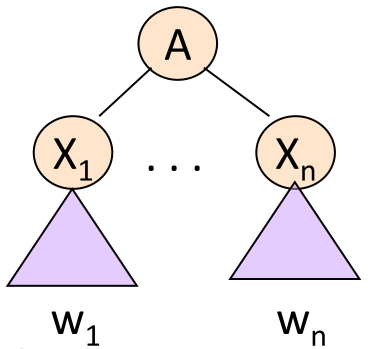


Часть 2 – Индукция

- IH: Предположим, что (2) выполняется для выводов за число шагов меньшее, чем k ,
- Пусть A $=>^*_{lm}$ w будет выводом за k шагов.
- Первый шаг есть A $=>_{lm}$ $X_1 \ldots X_n$.
- 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, $X_i =>^*_{lm} w_i$ для всех i таких, что X_i есть переменная и вывод имеет меньше, чем k шагов.
- По IH, 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

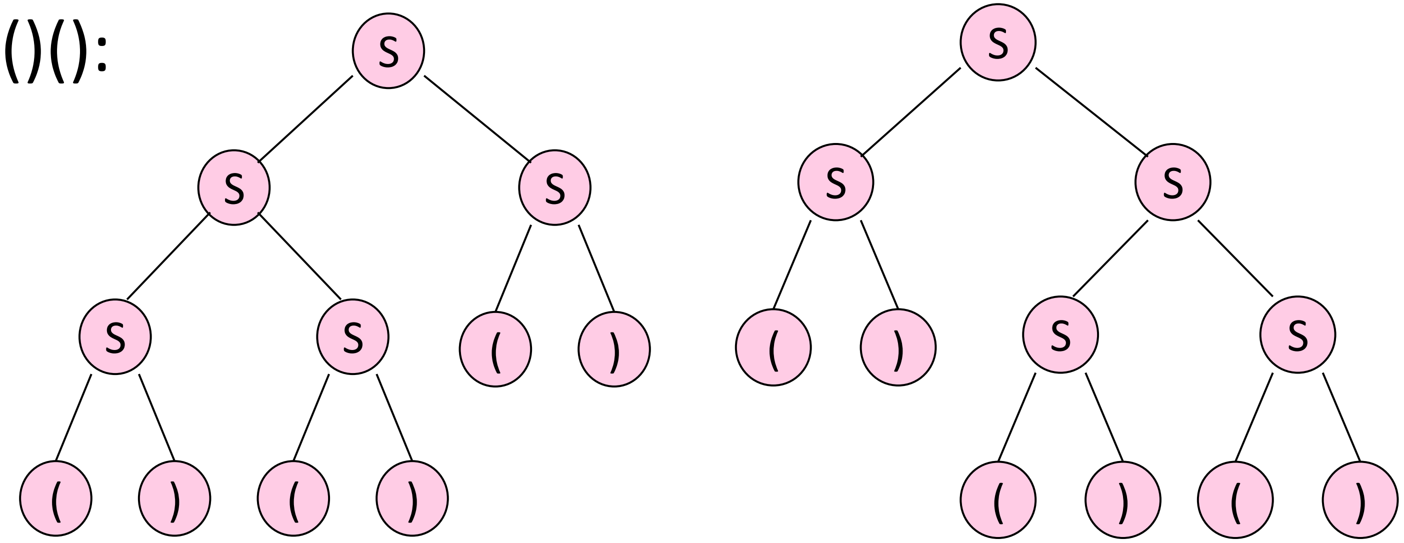
- Приведенные утверждения имеют зеркальное отражение для правых выводов.
- Если есть правый вывод w из A, то существует дерево разбора с корнем A и кроной w, и наоборот.
- Доказательство – аналогично.

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 $A \Rightarrow X_1 \ldots X_n$.
- И w все ещё можно разделить так, что первая часть выводится из X_1, следующая выводится из X_2, и.т.д.
- Таким образом, левые и правые варианты вывода могут быть смешаны.

Неоднозначные грамматики

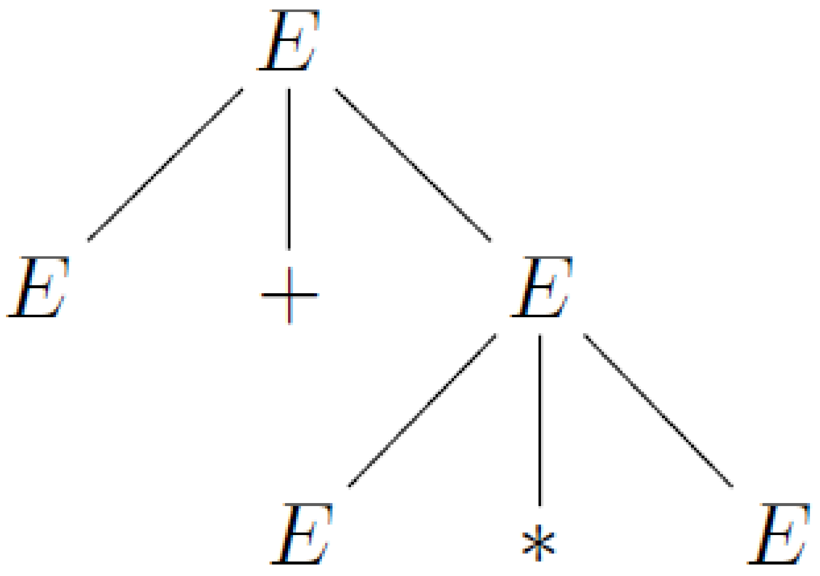
- **<u>Опр</u>**. КС-грамматика является *неоднозначной*, если существует строка языка, которая имеет два или более разных деревьев разбора.
- Пример: S -> SS | (S) | ()
- Два дерева разбора ()()():

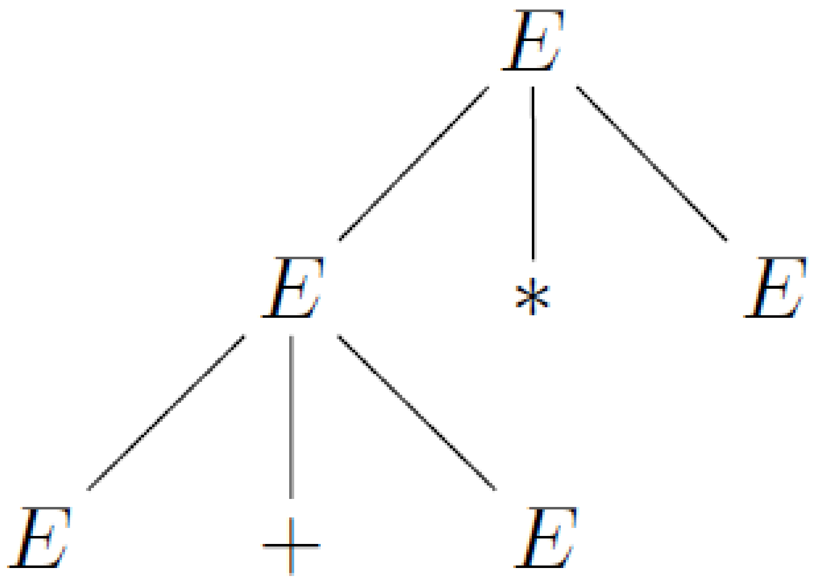


$$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, P, E),$$

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

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





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

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

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

$$\Longrightarrow$$

$$\begin{aligned} &E \to E+T \mid E*T \mid T \\ &T \to (E) \mid I \end{aligned}$$

$$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + I \Rightarrow a + b,$$
$$E \Rightarrow E + E \Rightarrow E + I \Rightarrow I + I \Rightarrow I + b \Rightarrow a + b.$$

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, начальный символ,
выводит сбалансированные строки.

B -> (RB | ϵ

R ->) | (RR

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 -> (RB | ϵ   R ->) | (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 -> (RB если следующий символ есть “(”; и используем ϵ если это конец.
  - Если нам нужно раскрыть R, используем R ->) если следующий символ есть “)” и (RR если это “(”.

Процесс разбора

Оставшийся вход:

(())()

Следующий символ

Шаги левого вывода:

B

B -> (RB | ϵ   R ->) | (RR

Процесс разбора

Оставшийся вход:

())()

Следующий символ

Шаги левого вывода

B

(RB

B -> (RB | ϵ   R ->) | (RR

Процесс разбора

Оставшийся вход:

))()

Следующий символ

Шаги левого вывода

B

(RB

((RRB

B -> (RB | є   R ->) | (RR

Процесс разбора

Оставшийся вход:

)()

Следующий символ

Шаги левого вывода

B

(RB

((RRB

(()RB

B -> (RB | ϵ   R ->) | (RR

Процесс разбора

Оставшийся вход:

()

Следующий символ

Шаги левого вывода:

B

(RB

((RRB

(()RB

(())B

B -> (RB | ϵ   R ->) | (RR

Процесс разбора

Оставшийся вход:

)

Следующий символ

Шаги левого вывода:

B

(RB

((RRB

(()RB

(())B

(())(RB

B -> (RB | ϵ R ->) | (RR

Процесс разбора

Оставшийся вход:

Следующий символ

Шаги левого вывода:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

B -> (RB | ϵ   R ->) | (RR

Процесс разбора

Оставшийся вход:

Следующий символ

Шаги левого вывода:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

(())()

B -> (RB | ϵ   R ->) | (RR

LL(1) - грамматики

- К слову, такие грамматики как
- B -> (RB | ϵ   R ->) | (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) - грамматиками.
  - “**L**eftmost derivation, **l**eft-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 $\{0^i1^j2^k \mid i = j \text{ или } j = k\}$ неоднозначный по своей природе.
- Интуитивно, по крайней мере, некоторые из строк вида $0^n1^n2^n$ 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 -> AB | CD

A -> 0A1 | 01

B -> 2B | 2

C -> 0C | 0

D -> 1D2 | 12

A генерирует равное число 0 и 1

B генерирует произвольное число 2-ек

C генерирует произвольное число 0-ей

D генерирует равное число 1 и 2

Существует два вывода каждой строки
С равным числом 0, 1, и 2. Например:
S => AB => 01B =>012
S => CD => 0D => 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.

- **<u>Теорема</u>**. Все языки типа 3 однозначны.
- <u>Доказательство</u>. Пусть L – язык типа 3 над алфавитом Σ.

Тогда существует КДА M=(Q, Σ,δ,q_0,F), такой что L(M)=L.

Но тогда L порождается грамматикой G=(Q, Σ,P, q_0), где

P={q$\to$ar | q,r$\in$Q, a$\in\Sigma$, $\delta(\mathrm{q,a})=\mathrm{r}$}$\cup${q$\to\varepsilon$ |q$\in$F}.

Очевидно, что G – однозначная.

Выводы

- Деревья разбора строки w $\in$L(G), левый и правый её выводы находятся во взаимном соответствии.
- Грамматика G наз. неоднозначной, если в определяемом ею языке L(G) существует строка, которая имеет два различных левых (правых) вывода.
- Неоднозначность – это свойство грамматики, а не языка.
- Если хотя бы одна грамматика языка L является однозначной, то L является *однозначным* языком.
- КС-язык L называется *существенно неоднозначным*, если все его грамматики неоднозначны.
- 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

Нормальные формы КС-грамматик

Удаление бесполезных переменных

Удаление ε-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*** (KC), если каждое правило из P имеет вид $A \to \beta$, 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.

КС-грамматики и ε-правила

- $A \to \varepsilon$ - это есть ε-правило.
- **Определение.** Назовем КС-грамматику $G = (N, \Sigma, P, S)$ грамматикой без ε-правил, если либо

1) P не содержит ε-правил, либо

2) есть точно одно ε-правило $S \to \varepsilon$ и S не встречается в правых частях правил из P.

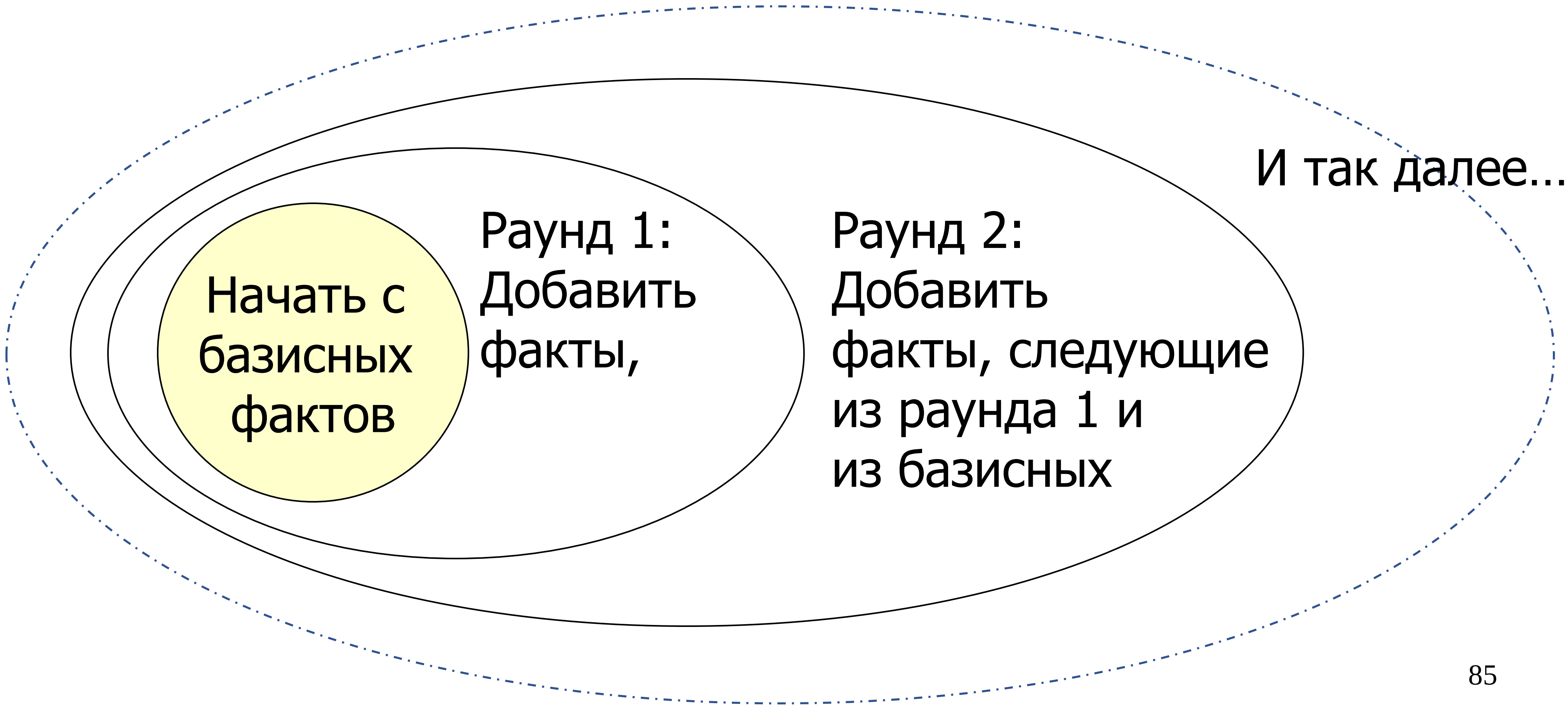
Переменные, из которых не выводится ничего

- Рассмотрим грамматику с правилами:
  - S -> AB,
  - A -> aA | a,
  - B -> 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

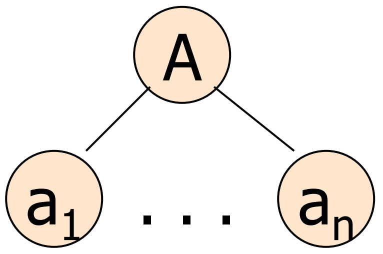


Проверка выводимости некоторой терминальной строки из переменной

- Базис: Если есть правило A => w, где w не имеет переменных, то A выводит терминальную строку w (из нетерминального символа A выводится терминальная цепочка символов).
- Индукция: Если есть правило A => α, где α 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

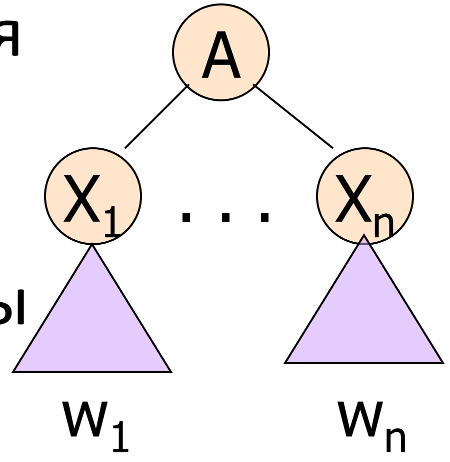
- Любая переменная, из которой выводится терминальная строка, может быть найдена приведенным алгоритмом.
- Доказательство представляет собой индукцию по высоте дерева синтаксического анализа наименьшей высоты, с помощью которого терминальная последовательность выводится из переменной A.
- Базис: Высота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



Базис алгоритма говорит нам, что переменная A будет найдена.

Индукция для док-ва обратного

- Пусть, IH выполняется для деревьев синтаксического анализа высоты < h, и предположим, что из A выводится терминальная строка посредством дерева разбора высоты h:
- По IH, X_i-ые – 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 -> AB | C,   A -> aA | a,   B -> bB,   C -> c

- Базис: A и C находятся, т.к. A -> a и C -> c.
- Индукция: S находится (помечается), т.к. S -> C.
- Больше ничего не находится.

Результат: S -> C,

A -> aA | a,

C -> 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 => α, то мы можем достичь α.

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

<u>Алгоритм</u> <u>УНС.</u> Устранение недостижимых символов.

<u>Вход</u>: КС-грамматика $G = (T, N, P, S\ )$

<u>Выход</u>: 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 α и β из $(N' \cup \Sigma'\ )^*$, что $S \Rightarrow^* \alpha X \beta$ в грамматике G'.

<u>Метод</u>.

1. Положить $V_0 = \{S\}$, и $\mathrm{i} = 1$.
2. Положить $V_i = \{X \mid \text{в } P \text{ есть } A \to \alpha X \beta \text{ и } A \in V_{i-1}\} \cup V_{i-1}$.
3. Если $V_i \neq V_{i-1}$, положить $\mathrm{i} = \mathrm{i} + 1$ 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 – важен порядок удаления

S -> AB,   A -> C,   C -> c,   B -> 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 -> bB, а затем и правило вывода из начального символа S -> AB.
- Затем, мы применяем алгоритм поиска недостижимых символов из начального и видим, что всё недостижимо, т.е. все правила будут удалены.
- Однако, если мы будем действовать неправильно и сначала будем удалять недостижимые символы из S, то увидим, что всё достижимо из S, т.е. ничего не удалиться.
- Затем, когда мы будем искать символы, которые не порождают терминальных строк, мы исключаем только B.
- Т.о. A, C, и c никогда не будут удалены, останутся правила A -> C, C -> 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)$\neq \varnothing$ и «НЕТ» – в противном случае.

Метод. Рекурсивно строим множества $N_0, N_1, \ldots$

1. $N_0 = \emptyset, \mathrm{i} = 1$
2. $N_i = \{A \mid A \to \alpha \in P \text{ и } \alpha \in (N_{i-1} \cup T)^*\} \cup N_{i-1}$
3. Если $N_i \neq N_{i-1}$, то , $\mathrm{i} = \mathrm{i} + 1$ 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'∪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

- **<u>Теорема</u>**. Грамматика G’, которую строит алгоритм «УБС» не содержит бесполезных символов и L(G’) = L(G) .
- <u>Доказательство</u>.
- 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.

ε-правила

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

ε-правила

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

Пустые символы

- Чтобы удалить є-правила, нужно сначала найти *пустые символы* = переменные A такие, что A =>* є.
- Базис: Если есть правило A -> є, то A пустой.
- Индукция: Если есть правило A -> α, и все символы в α пустые, то A пустой.

Пример: пустые символы

S -> AB,

A -> aA | є,

B -> bB | A

- Базис: A – пустой, т.к. A -> є.
- Индукция: B – пустой, т.к. B -> A.
- Далее, S – пустой, т.к. S -> AB.

Удаление ϵ-правил

- Идея: преобразовать каждое правило A -> $X_1 \dots X_n$ 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 ϵ.

Удаление є-правил

Алгоритм УєП. Преобразование в грамматику без є-правил.

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

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

Метод.

1. Построить $N_e = \{A | A \in N \text{ и } A \Rightarrow_G^+ \varepsilon\}$
2. Построить P’:

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

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

- Если $S \in N_e$, включить в P’ правила S’$\longrightarrow \varepsilon$|S, где S’ – новый начальный нетерминал и положить N’=N$\cup${S’}. В противном случае положить N’=N и S’=S.

3. Определить новую $G' = (T', N,' P', S\ ')$. Очевидно, что G' эквивалентна G'.

Пример

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

Пример: Удаление ϵ-правил

S -> ABC,   A -> aA | ϵ,   B -> bB | ϵ,   C -> ϵ

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

S -> ABC | AB | AC | BC | A | B | C

A -> aA | a

B -> bB | b

S’ -> S | ϵ

Замечание: C - теперь бесполезный.

Пример: Удаление ϵ-правил

S -> ABC, A -> aA | ϵ, B -> bB | ϵ, C -> ϵ

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

S -> ~~ABC~~ | AB | ~~AC~~ | ~~BC~~ | A | B | ~~C~~

A -> aA | a

B -> bB | b

S' –> S | ϵ

Замечание: C - теперь бесполезный. Удаляем его правила.

Почему это работает

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

Док-во 1 – Базис

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

Док-во 1 – Индукция

- Пусть A $=>*_{old}$ w есть вывод за k шагов, и предположим, что IH выполняется для выводов длиной меньше, чем k.
- Пусть первым шагом вывода будет A $=>_{old}$ $X_1 \ldots X_n$.
- Тогда w можно разбить: $w = w_1 \ldots w_n$, где для всех i,

w_i - часть w , которая либо есть X_i (если X_i- терминал), либо

$X_i =>*_{old} w_i$, за меньшее, чем k шагов.

Индукция – Продолжение

- Если X_i- переменная и $w_i \neq \epsilon$, то по IH, $X_i =>^*_{new} w_i$.
- 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 ϵ, потому что $w \neq \epsilon$.
- При использовании этого правила в новой грамматике по выводам $X_i =>^*_{new} w_i$, 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

- **<u>Опр</u>**. *Единичные правила* - это те правила, в теле которых есть только одна переменная.
- Эти правила могут быть удалены.
- Ключевая идея: Если A =>* B в результате серии единичных выводов, и B -> α неединичное правило, то добавим правило A -> α.
- Затем, удалим все единичные правила.

Единичные правила – (2)

- Алгоритм.
- Найдем все пары (A, B) такие, что A =>* 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 -> 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 =>* B по единичным правилам.
- Обратно, индукцией по числу шагов в выводе единичными правилами A =>* B, мы можем показать, что пара (A, B) найдена.

Алгоритм устранения единичных правил

- **<u>Алгоритм</u> УЕП**.
- <u>Вход</u>. КС-грамматика без ε-правил.
- <u>Выход</u>. Эквивалентная КС-грамматика G’ без ε-правил и без единичных правил.
- <u>Метод</u>. 1. Для каждого $A \in N$ построить $N_A = \{B | A \Longrightarrow^* B\}$ следующим образом:

(а) Положить $N_0 = \{A\}$ и i=1.

(б) Положить $N_i = \{C | B \longrightarrow C \in P \text{ и } B \in N_{i-1}\} \cup N_{i-1}$.

(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 $B \longrightarrow \alpha \in P$ и не является единичным правилом, включить в P’ правило $A \longrightarrow \alpha$ для всех таких A, что $B \in N_A$.

3. Положить $G' = (N, \Sigma, P', S)$.

Доказательство, что алгоритм удаления единичных правил работает

- <u>Теорема</u>. Алгоритм УЕП строит грамматику G’ без единичных правил и L(G’)=L(G).
- Основная идея: Вывод A $=>^*_{lm}$ 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 – {ϵ} такая, что в ней:
  1. Нет бесполезных символов.
  2. Нет ϵ-правил.
  3. Нет единичных правил.
- Т.е., каждое тело правил есть либо единичный терминал, либо имеет длину ≥ 2.
- Такого рода КС-грамматика называется «приведенной».

Очистка грамматики– (2)

- Док-во: Начнём с КС-грамматики для L.
- Выполним следующие шаги:
  1. Удалим ϵ-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 -> BC (тело имеет две переменные).
  2. A -> a (тело – единичный терминал).
- Теорема: Если L – КС-язык, то L – {ϵ} имеет КС-грамматику в НФХ.

Одно из приложений НФХ

- Одним из важных применений приведения грамматик в НФХ является то, что это дает нам относительно эффективный алгоритм проверки членства строки в КС-языке.
- Такой тест можно было бы легко сделать, глядя на все выводы определенной ограниченной длины,
- С ε-правилами и единичными правилами в грамматике, не очевидно, насколько длинным будет вывод даже короткой терминальной строки. Более того, даже если бы мы могли ограничить длину вывода- (как мы можем) - мы все равно столкнулись бы с экспоненциальным алгоритмом по длине конечной строки алгоритмом.
- Преобразовав грамматику в НФХ, мы можем сделать этот тест максимум кубическим от длины строки.

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 (тела) правила.
  - Для каждого терминала *a* создаём новую переменную A_a и правило A_a -> a.
  - Заменяем *a* на A_a в теле правил длиной $\geq$ 2.

Пример: Шаг 2

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

  A -> BF, F -> CG, и G -> DE.

  - F и G должны использоваться везде.

Пример шага 3 – продолжение

- Итак, A -> BCDE заменяется на A -> BF, F -> CG, и G -> DE.
- В новой грамматике, A => BF => BCG => 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 $G=(N,\Sigma,P,S)$

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

S -> SS | 1A0
A -> 1A0
A -> ε

Удаляем ε-правила:
S-> SS | 1A0 | 10
A-> 1A0 | 10

цепные:
нет таких

бесполезные:
нет таких

S-> SS|EAO|EO
E-> 1
O-> 0
A-> EAO|EO

S-> SS|ER|EO
R-> AO
E-> 1
O-> 0
A-> ER|EO

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

$$S \to AB, A \to SA, A \to BB, A \to bB, B \to b, B \to aA, B \to \epsilon$$

S -> AB
A- > SA | BB | bB
B -> b | aA | $\boldsymbol{\varepsilon}$

1. удаляем ε-правила
2. удаляем цепные (единичные) правила
3. удаляем бесполезные символы

1. удаляем $\boldsymbol{\varepsilon}$-правила
S-> A|AB|B
A-> SA|BB|B|bB|b|S|A
B-> b|aA|a

2. удаляем цепные правила
(S,S) & S-> A (S,A)
(S,A) & A-> B (S,B)
(A, A) & A-> B (A,B)

(S,S) S-> AB
(S,A) S-> SA|BB|bB|b
(S,B) S-> b|aA|a

S-> AB|SA|BB|bB|b|aA|a
A-> SA|BB|bB|b|aA|a
B-> b|aA|a

3. удаляем бесполезные
таких нет

<u>Приведение к НФХ</u>

S-> AB|SA|BB|KB|b|RA|a
K-> b
R->a
A-> SA|BB|KB|b|RA|a
B-> b|RA|a

Автоматы с магазинной памятью

Определение
Функционирование МП-автомата
Языки для МП-автоматов
Детерминированные МП-автоматы

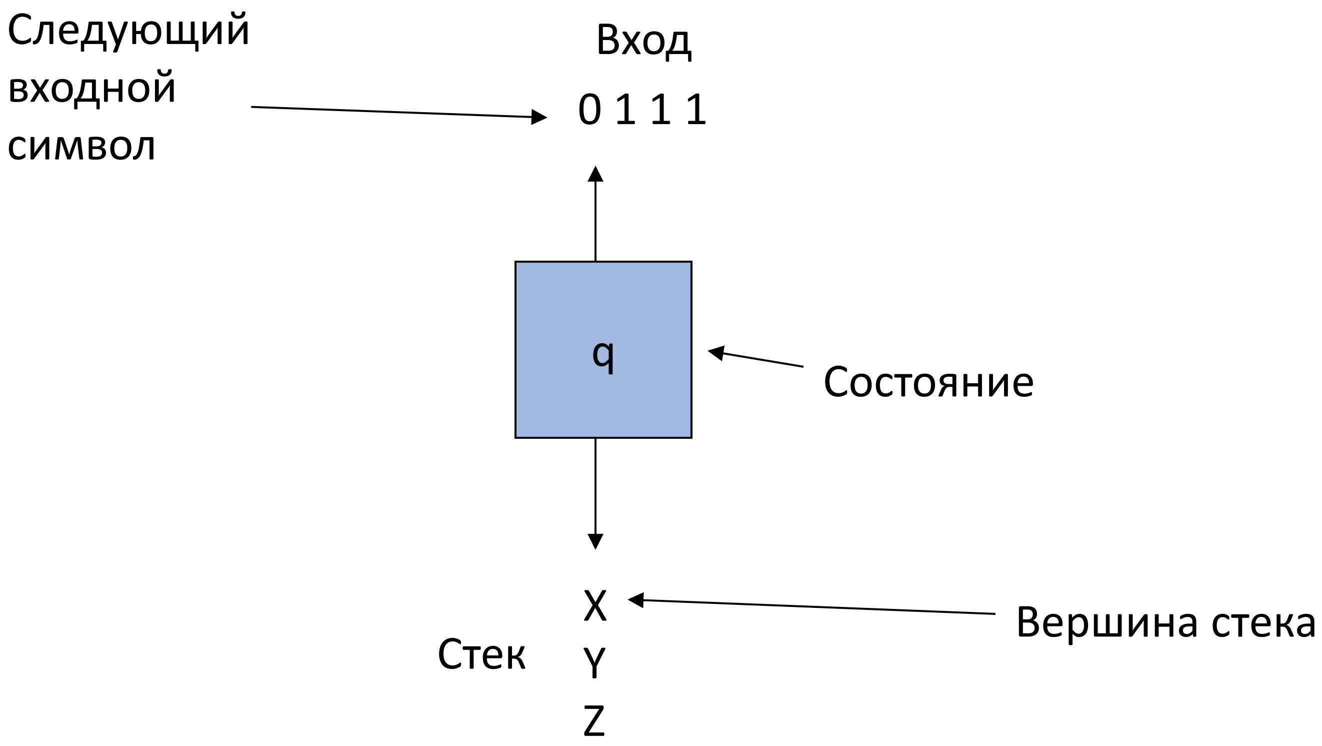
Автоматы с магазинной памятью

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

Неформально: МПА

- Вообразим ϵ-НКА с дополнительной возможностью манипулирования с памятью в виде стека.
- Функционирование МПА определяется следующими параметрами:
  1. Текущее состояния (его “НКА”),
  2. Текущий входной символ (или ϵ), и
  3. Текущий символ вершины стека.

Схема МПА



Следующий входной символ
Вход
0 1 1 1
q
Состояние
X
Y
Z
Стек
Вершина стека

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, Σ, Γ, δ, q_0 , Z_0, F)
  1. Конечным множеством *состояний* (Q, обычно).
  2. *Входным алфавитом* (Σ, обычно).
  3. *Алфавитом стека* (Γ, обычно).
  4. *Функцией переходов* (δ, обычно).
  5. *Начальным состоянием* (q_0, в Q, обычно).
  6. *Начальным символом* (Z_0, в Γ, обычно).
  7. Множеством *конечных состояний* ($F \subseteq Q$, обычно).

Соглашения

- a, b, … - входные символы.
  - Но иногда мы позволяем использовать є в качестве возможного значения.
- …, X, Y, Z – символы стека.
- …, w, x, y, z – строки входных символов
- α, β,… - строки стековых символов.

Функция переходов

- Имеет три аргумента:
  1. Состояние, в Q.
  2. Вход, который является либо символом в Σ или ϵ.
  3. Верхний стековый символ в Γ.
- δ(q, a, Z) – есть множество нуль или более действий вида (p, α).
  - p – новое состояние; α - строка стековых символов, которая заменяет символ вершины стека.

Действия МПА

- Если $\delta(q, a, Z)$ содержит (p, α) 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 ϵ).
  3. Заменить Z в вершине стека на α.
- МПА может иметь несколько альтернативных пар вида (p, α) для $\delta(q, a, Z)$.

Пример: МПА

- Построим МПА, принимающий $\{0^n1^n \mid n \geq 1\}$.
- 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.

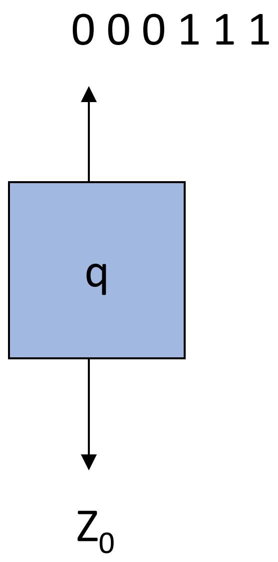
Пример: МПА – (2)

- 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.

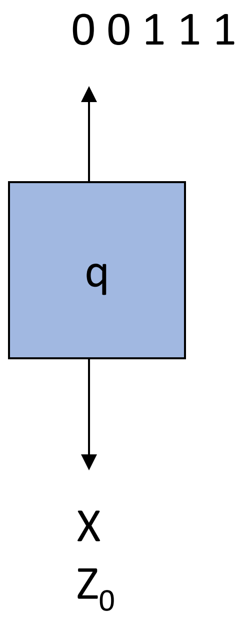
Пример: МПА – (3)

- 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.

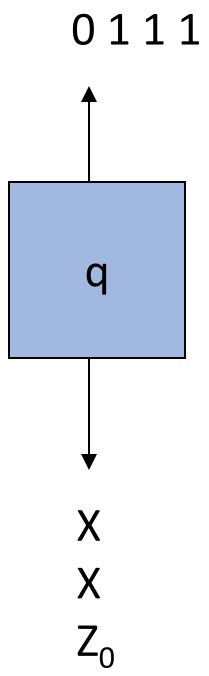
Действия: Пример МПА



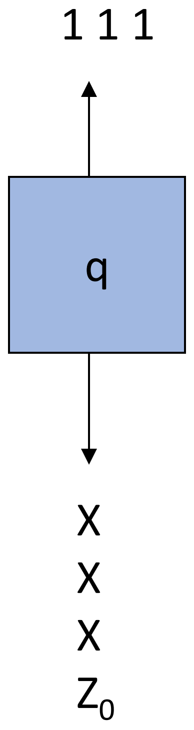
Действия: Пример МПА



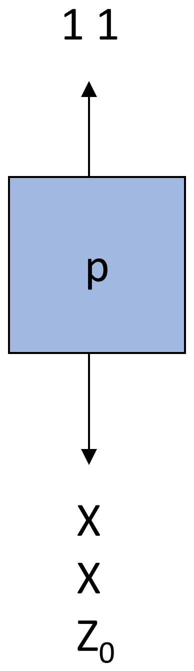
Действия: Пример МПА



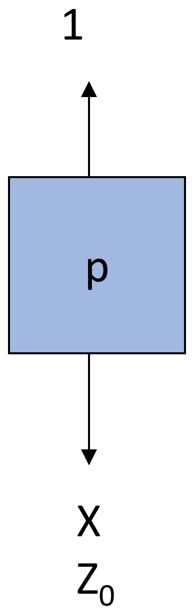
Действия: Пример МПА



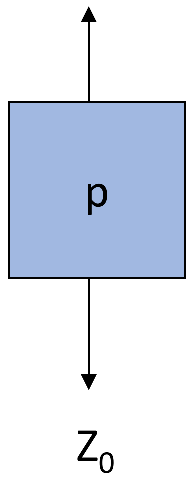
Действия: Пример МПА



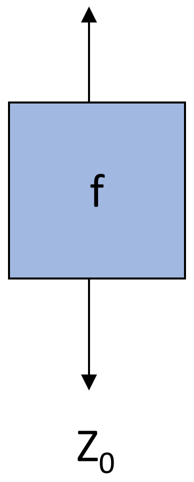
Действия: Пример МПА



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, α), где:

1. q – текущее состояние.
2. w – оставшийся непрочтенный вход.
3. α - 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 α, если $\delta(q, a, X)$ содержит (p, β).
- 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 α.

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) -> 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) -> 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 любогоX.

Доказательство: N(P) -> 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) -> 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 более одного выбора при переходе из любого состояния q, для входного символа *a* (включая ϵ), и стекового символа X.
- Дополнительно, не должно быть выбора между использованием входа ϵ 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 свой 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 сделать 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 P

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

От МПА к КСГ

- Теперь предположим, что L = N(P).
- <u>**Теорема**</u>. 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

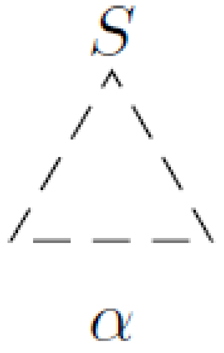
Интуитивно

- 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.

- **<u>Лемма</u>**. (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

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

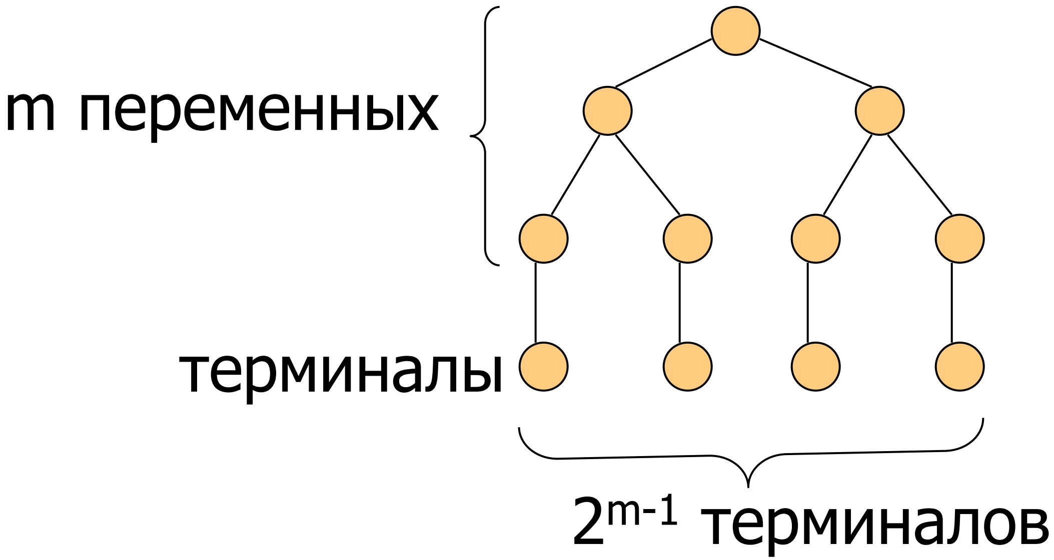
1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. Для всех $i \geq 0$, uv^iwx^iy принадлежит L.

Доказательство леммы о накачке

- Начнём с грамматики в НФХ для $L - \{\epsilon\}$.
- Пусть грамматика имеет m переменных (нетерминалов).
- Возьмём $n = 2^m$.
- Пусть слово z длины $\geq n$, будет в L.
- Докажем, что (“см. пред. *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 $\leq$ m+1, то набольшей кроной будет строка длины 2^{m-1}, как в:

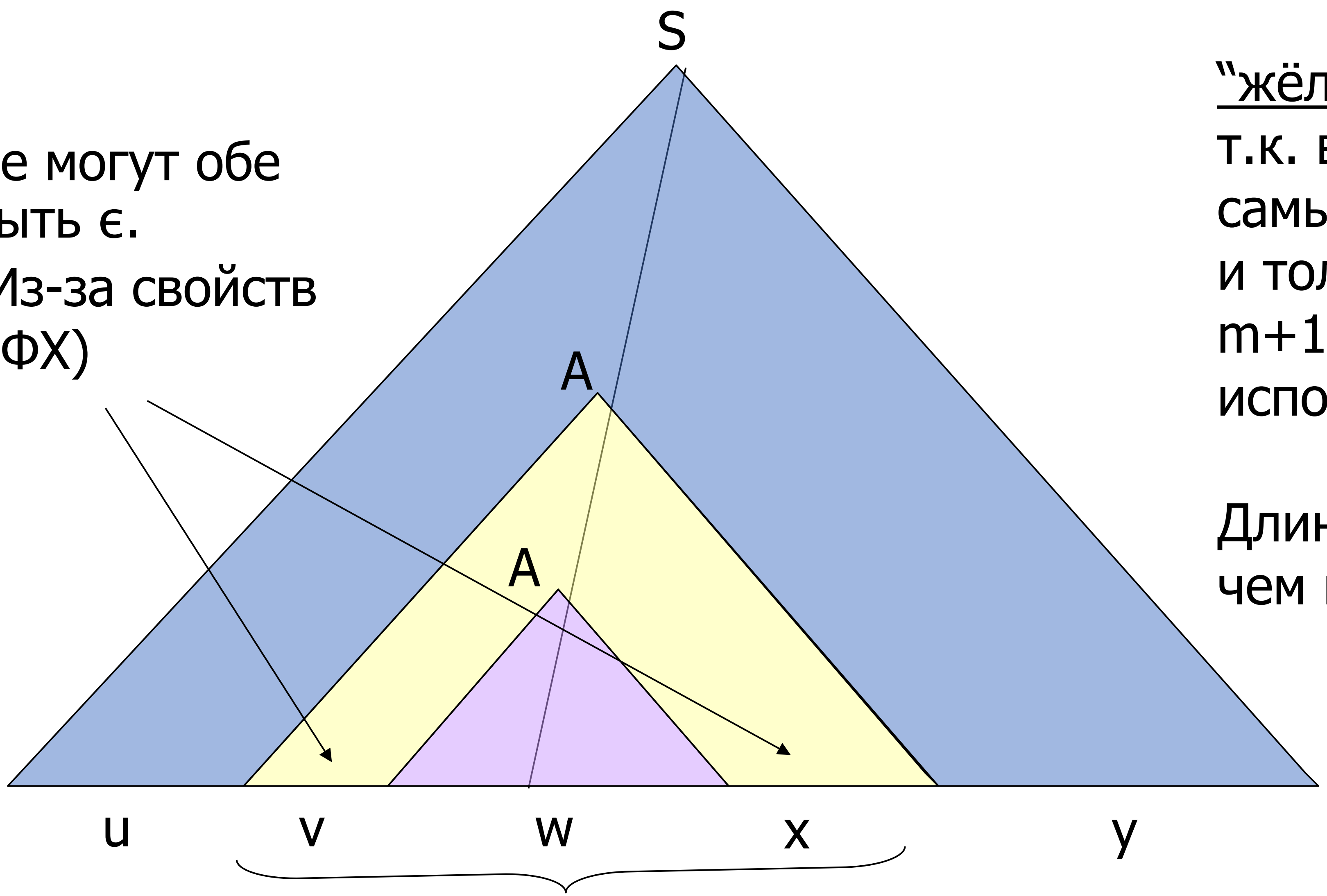


Возврат к доказательству леммы о накачке

- 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

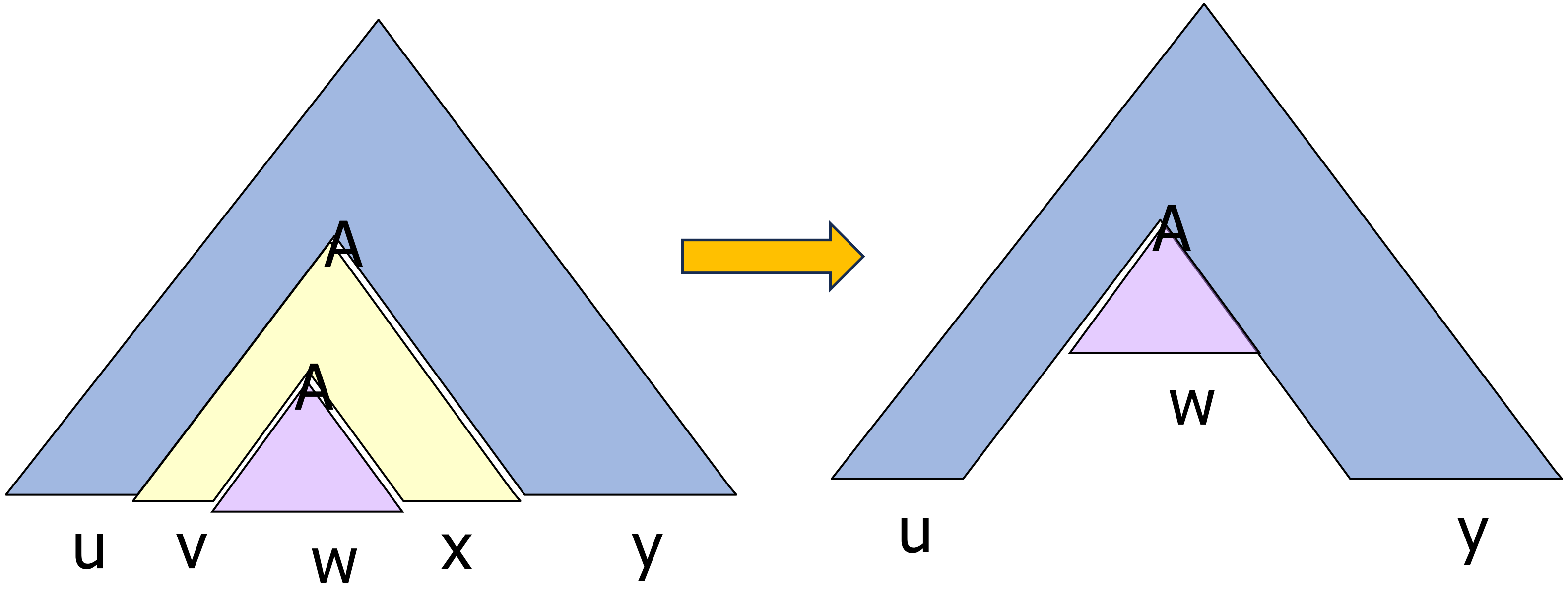
Не могут обе быть ϵ.
(Из-за свойств НФХ)



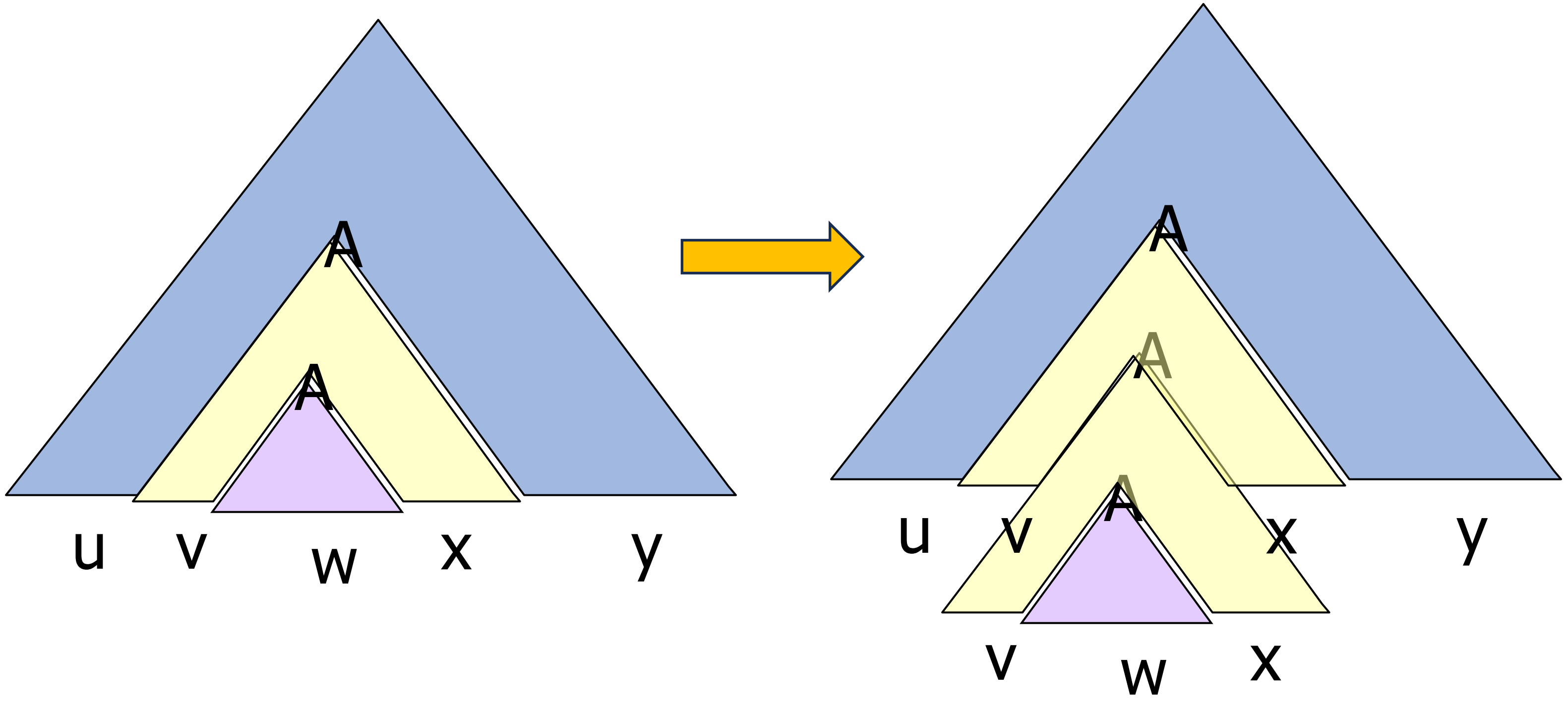
“жёлтый”+”фиол.”< $2^m = n$
т.к. выбран
самый длинный путь
и только нижние
m+1 переменных
использовано.

Длина vwx не более, чем n.

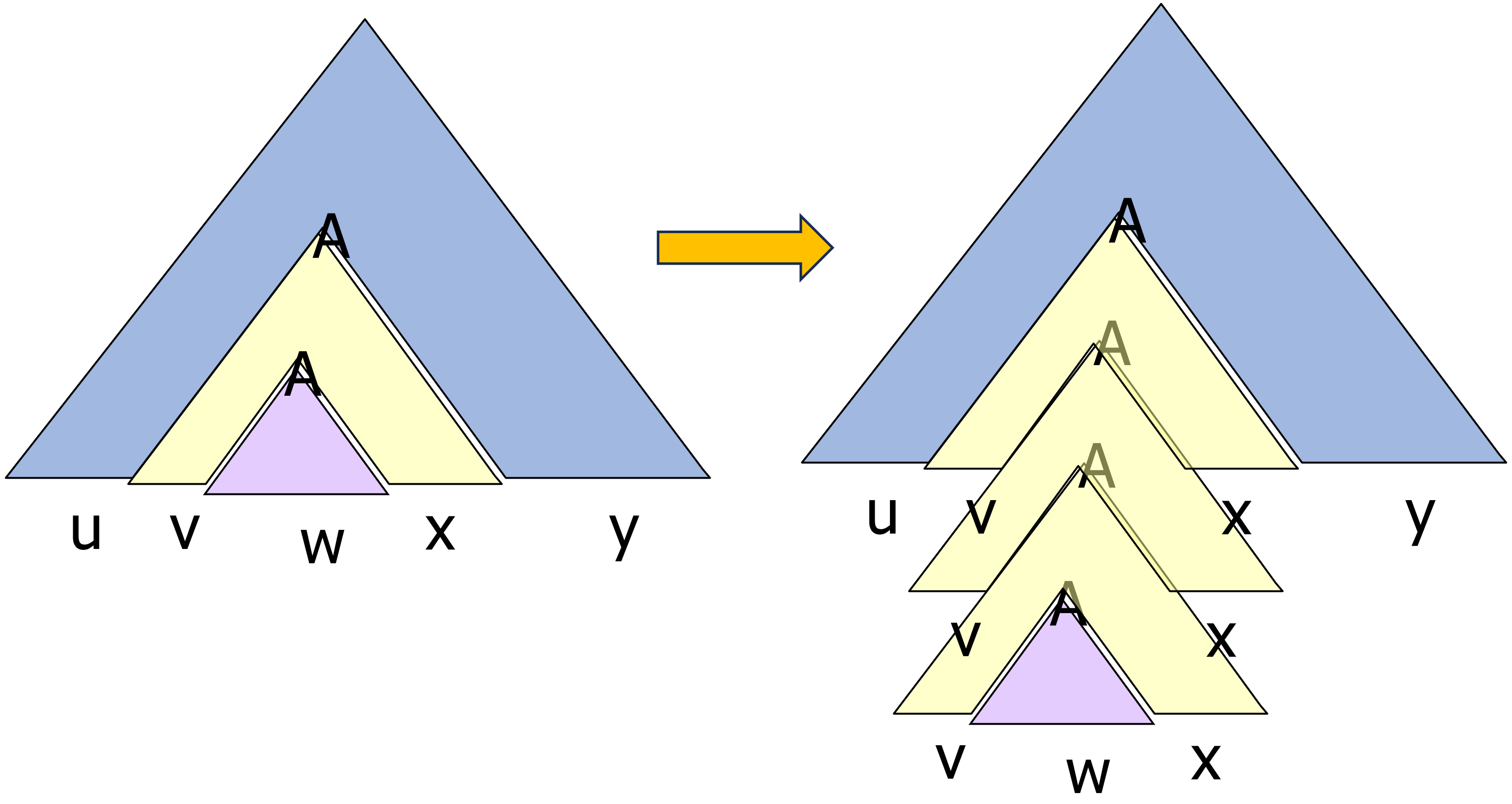
Накачка нуль раз



Накачка два раза



Накачка три раза и т.д.



Использование леммы о накачке

- $\{0^i10^i \mid i \geq 1\}$ есть КС-язык.
  - Мы можем отметить одну пару счетчиков.
- Но $L = \{0^i10^i10^i \mid i \geq 1\}$ - нет.
  - Мы не можем выделить две пары счётчиков, или счетчик трёх компонентов 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 строку $z = 0^n10^n10^n$.
- Мы можем записать $z = uvwxy$, 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 uwy имеет, по крайней мере, одну 1, что при накачке не даёт строку из L.

Использование леммы о накачке – (3)

- Мы все еще рассматриваем $z = 0^n10^n10^n$.
- Случай 2: vx имеет, по крайней мере, один 0.
  - vwx слишком коротка (длина $\leq$ n), чтобы расширить все три блока 0-ей в $0^n10^n10^n$.
  - Рассмотрим строку uwy, которая, если L-КС-язык, должна быть в L.
  - НО, uwy имеет, по крайней мере, один блок из n 0-ей, и, по крайней мере, один блок с меньшим, чем n 0-ями.
  - Таким образом, uwy 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 = ϵ является специальным случаем, разрешаемым посредством проверки того, является ли начальный символ пустым.
- Алгоритм (*CYK*) является хорошим примером динамического программирования и работает время порядка $O(n^3)$, где $n = |w|$.

CYK =John Cocke, Dan Younger, and Tadao Kasami

CYK алгоритм

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

CYK алгоритм – (2)

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

Пример: CYK алгоритм

Грамматика: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

Строка w = ababa

K=1 : $X_{12}=\{B,S\}$ $X_{23}=\{A\}$ $X_{34}=\{B,S\}$ $X_{45}=\{A\}$

$X_{11}=\{A,C\}$ $X_{22}=\{B,C\}$ $X_{33}=\{A,C\}$ $X_{44}=\{B,C\}$ $X_{55}=\{A,C\}$

Пример: CYK алгоритм

Грамматика: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

Строка w = ababa

X_{13}={} Ничего не дает

K=1 :

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}

Пример: CYK алгоритм

Грамматика: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

Строка w = ababa

K=2 :

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}

Пример: CYK алгоритм

Грамматика: S -> AB, A -> BC | a, B -> AC | b, C -> a | b
Строка w = ababa

K=1 :

K=3 :

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

$X_{13}=\{A\}$ $X_{24}=\{B,S\}$ $X_{35}=\{A\}$

$X_{12}=\{B,S\}$ $X_{23}=\{A\}$ $X_{34}=\{B,S\}$ $X_{45}=\{A\}$

$X_{11}=\{A,C\}$ $X_{22}=\{B,C\}$ $X_{33}=\{A,C\}$ $X_{44}=\{B,C\}$ $X_{55}=\{A,C\}$

Пример: CYK алгоритм

Грамматика: S -> AB, A -> BC | a, B -> AC | b, C -> a | b

Строка w = ababa

X_{15}={A}

Т.к. S нет в X_15, мы заключаем, что строки ababa нет в языке данной грамматики.

K=4 :

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}

Проверка бесконечности

- Пусть L=L(G). |L| = +$\propto$?
- 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.
- Что нужно взять за n?

Проверка бесконечности

- Пусть $L=L(G)$. $|L| = +\propto$?
- 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.
- Что нужно взять за n?
- $n = 2^m$
- где m – число переменных (нетерминалов) грамматики.

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)

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

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)

- Образуем новую грамматику для LM, начав с того, что включим в неё все символы и продукции G и H.
- Добавим новый начальный символ S.
- Добавим продукцию S -> 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.
- Образуем новую грамматику для L* путём введения в G нового начального символа S и продукций $S \to S_1S \mid \epsilon$.
- Правый вывод из S генерирует последовательность из нуль 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 -> 0S1 | 01.
  - Обращение языка L(G) имеет грамматику S -> 1S0 | 10.
- Доказательство проводится индукцией по длине вывода в двух грамматиках.

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).

- *Гомоморфизмом* на алфавите является функция, которая каждый символ этого алфавита заменяет некоторой строкой.
- Пример: h(0) = ab; h(1) = ϵ.
- Расширим функцию на строки $h(a_1 \ldots a_n) = h(a_1) \ldots h(a_n)$.
- Пример: h(01010) = ababab.

Пример: Замыкание КС-языков по гомоморфизму

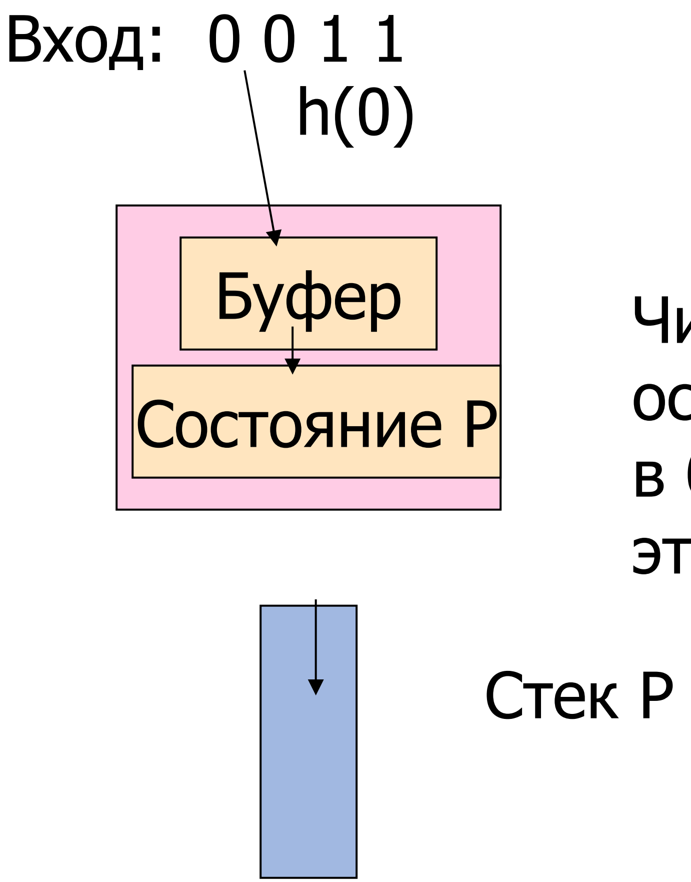
- G имеет продукции S -> 0S1 | 01.
- h определяется как h(0) = ab, h(1) = ϵ.
- h(L(G)) имеет грамматику с правилами S -> abS | ab.

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 $h^{-1}(L)$.
- Идея: P’ моделирует P,
- P’ должен применять h к каждому обозреваемому входному символу.
- P’ имеет двукомпонентные состояния: 1-ая компонента – состояние P, 2-ая – буфер, который содержит суффикс того, что вы получаете, применяя h к какому-то символу.
- Этот буфер позволяет P’ использовать символы h(a) один за такт, чтобы производить такты работы P.

Архитектура P’

P' может считать свой первый входной символ, 0, и применить к нему h. Буфер, который изначально был пустым, теперь имеет строку h(0). Это может быть длинная строка, но ее длина конечна, поэтому в P' может быть только конечное число состояний.



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

Теперь, чтобы смоделировать P, P' принимает первый символ h(0) и имитирует P, используя то, что является следующим входным символом. Это моделирование может занять много тактов, так как могут быть переходы по входному эпсилон, а также один переход на самом символе. Однако символ удаляется из передней части буфера, поэтому в следующий раз, когда P нужен реальный входной символ, он получает второй символ h(0). Моделирование продолжается таким образом до тех пор, пока все символы h(0) не будут удалены из буфера. В этот момент P' может применить h к своему следующему входу и перегрузить буфер.

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, ϵ], для которых q является финальным состоянием P.

Переходы P’

1. δ’([q, ϵ], 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. δ’([q, bw], ϵ, X) содержит ([p, w], α), если δ(q, b, X) содержит (p, α), где b есть либо входной символ P, либо ϵ.
    - 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 -> AB, A -> 0A1 | 01, B -> 2B | 2.
- Это так и для $L_3 = \{0^i 1^n 2^n \mid n \geq 1, i \geq 1\}$.
- Но $L_1 = L_2 \cap L_3$!

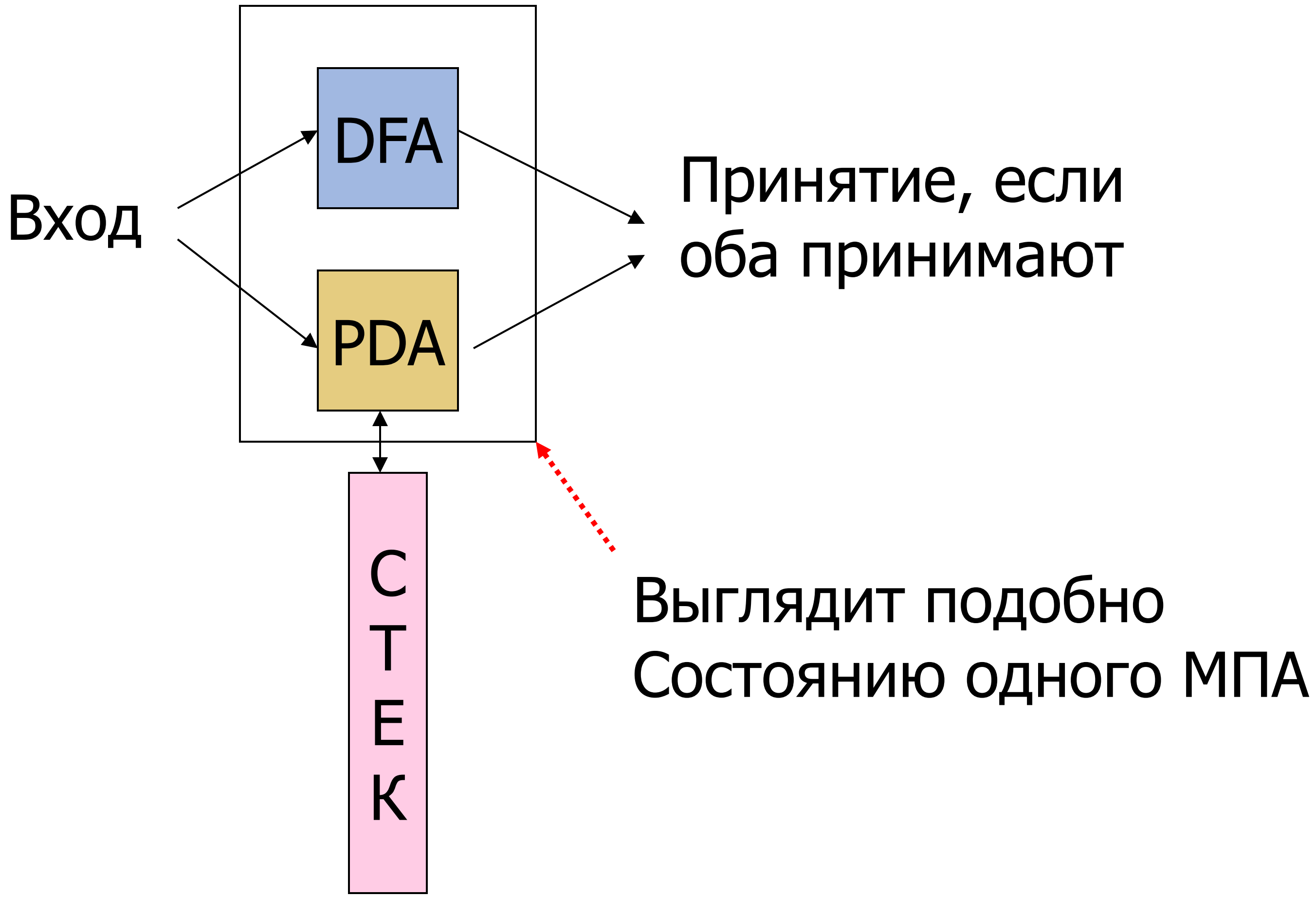
Отсутствие замыкания по разности

- 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

- Итак, 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, имея в виду, что их комбинация является МПА.
  - МПА принимает по переходу в заключительное состояние.

КДА и МПА в параллели

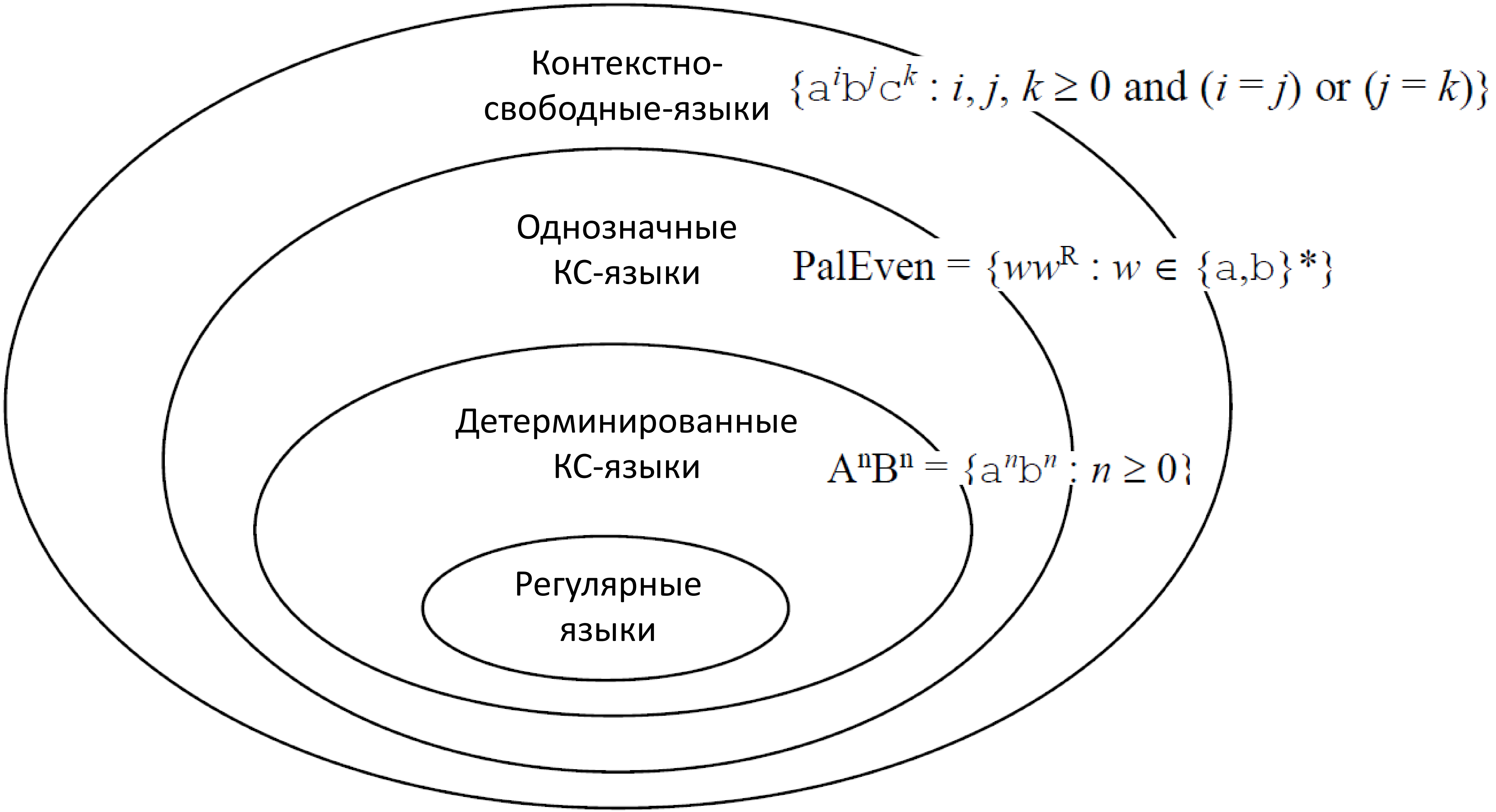


Формальное построение

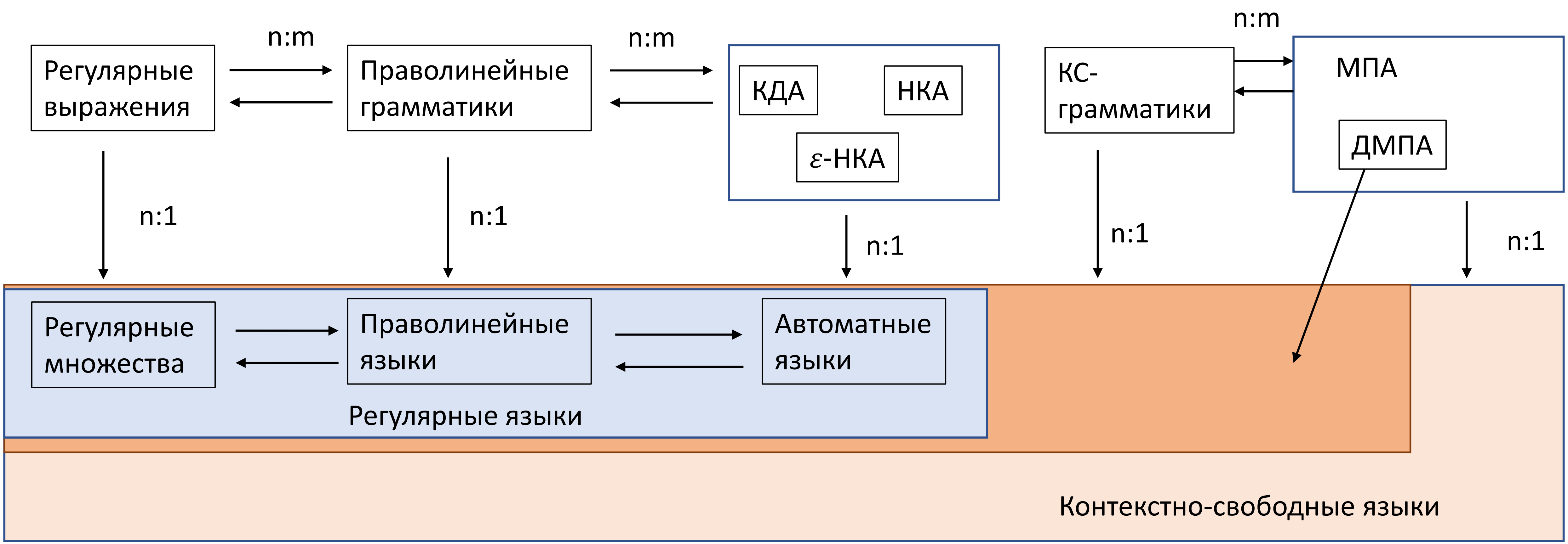
- Пусть КДА A имеет функцию переходов δ_A.
- Пусть МПА P имеет функцию переходов δ_P.
- Состояниями комбинированного МПА являются [q,p], где q – состояние A и p - состояние P.
- $\delta([q,p], a, X)$ содержит $([\delta_A(q,a),r], \alpha)$ если $\delta_P(p, a, X)$ содержит (r, α).
  - Заметим, что в качестве a может быть ε, 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: $(p_0, w, Z_0)\vdash^*(p, \varepsilon, \alpha)$.


Контекстно-
свободные-языки
$\{a^i b^j c^k : i, j, k \geq 0 \text{ and } (i = j) \text{ or } (j = k)\}$
Однозначные
КС-языки
$\mathrm{PalEven} = \{ww^R : w \in \{a,b\}^*\}$
Детерминированные
КС-языки
$\mathrm{A^nB^n} = \{a^n b^n : n \geq 0\}$
Регулярные
языки

Подведём итоги



Разрешимые свойства для регулярных языков

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

Разрешимые свойства для КС-языков 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: $L \cap M=\varnothing$?

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 являются очень важным инструментом программирования.
- Но на уровне представления данных, есть только один тип, который можно понимать как целый или строковый.
- Ключевая идея: Строки из 0 и 1, которые представляют программы, это только другой способ думать об одном и том же типе данных. Т.е., программы – это целые числа

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».
- Исправить это можно 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.

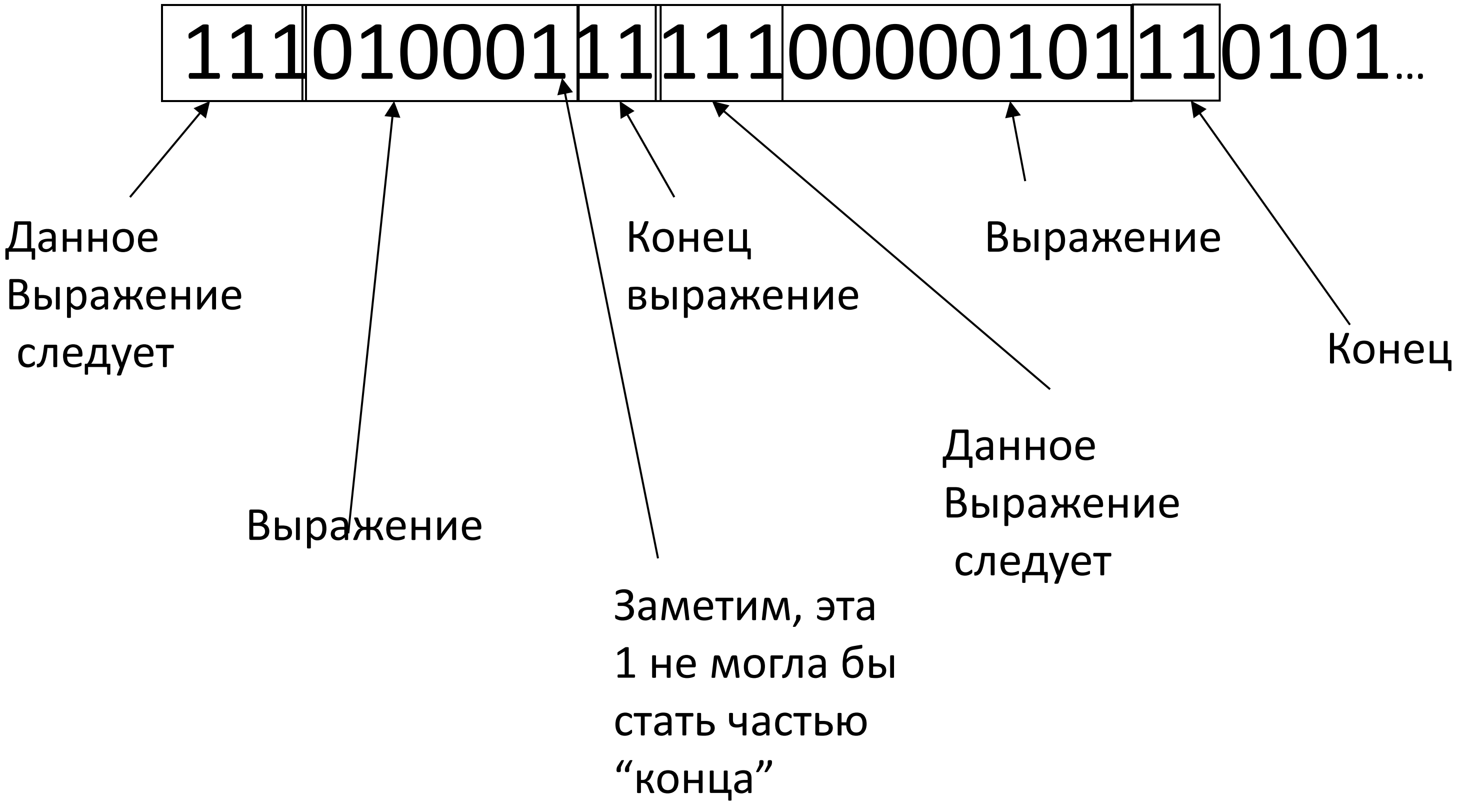
Доказательства – (2)

- 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

- Конечное множество – это множество, у которого невозможно найти 1-1 отображение между элементами множества и любым собственным подмножеством самого множества.
- *Конечные множества* имеют в качестве характеристики особые числа, которые являются числом элементов в множестве.
- Пример: {a, b, c} – конечное множество; его *кардинал* есть 3.

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,…} есть бесконечное множество.
  - Существует 1-1 соответствие 1<->2, 2<->4, 3<->6,… между этим множеством и соответствующим подмножеством (множеством четных чисел).

Счётные множества

- *Счётные множества* есть множества с 1-1 соответствие с целыми положительными целыми числами.
  - Следовательно, все счётные множества – бесконечные.
- Пример: Все целые.
  - 0<->1; -i <-> 2i; +i <-> 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],…, [1,4], [5,1],…
- Интересное применение: опишем функцию f(i,j) такую, что пара [i,j] соответствует целому f(i,j) в этом порядке. Таким образом, можно говорить об i-ой паре чисел.

Перечисления

- *Перечисление* множества - это 1-1 соответствие между 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 | w - i-ая бинарная строка и w не находится в i-м языке}.

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.

L_j

x

Вспомним: L = { w | w есть i-ая бинарная строка и w не принадлежит i-ому языку}.

j-th

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 строки в языке.

Рисунок диагонализации

Процесс создания языка L, который не может быть ни в одном перечислении, называется диагонализацией».
M[i,j]=1 означает, что j-ая строка содержится в i-м языке.
Рассмотрим диагональ и инвертируем ее значения.

Строки

Языки

	1	2	3	4	5	…
1	1	0	1	1	0	…
2		1				
3			0			
4				0		
5					1	
…						…

Рисунок диагонализации

Инвертируем каждый диагональный элемент

Строки

Языки	1	2	3	4	5	…
1	0	0	1	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…

Не может быть строкой (языком) – Это не согласуется с элементами каждой строки.

Этот язык не может быть ни одной строкой, потому что он не согласуется с каждой строкой хотя бы в одной позиции.

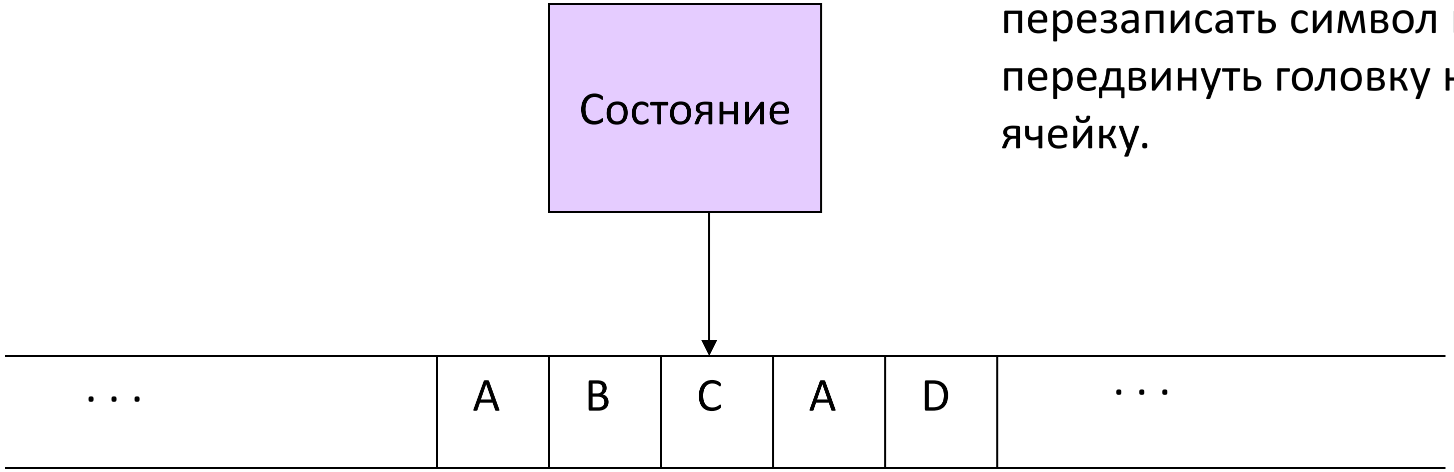
В частности, он не согласуется с i-м рядом в i-й позиции.

Теория машин Тьюринга

- Цель теории машин Тьюринга – доказать, что некоторые специфические языки не имеют алгоритмов (задания/распознавания).
- Начинают с определения 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 символы из некоторого конечного алфавита

Формальное определение МТ

- ТМ определяется:
  1. Конечным множеством *состояний* (Q, обычно).
  2. *Входной алфавит* (Σ, обычно).
  3. *Алфавит ленты* (Γ, обычно; содержит Σ).
  4. *Функция переходов* (δ, обычно).
  5. *Начальный символ* (q_0, в Q, обычно).
  6. *Пустой символ* (B, в $\Gamma - \Sigma$, 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 Г.
- δ(q, Z) - либо не определена, либо является тройкой вида (p, Y, D).
  - p - состояние.
  - Y – новый символ ленты.
  - D – *направление сдвига головки на ленте*, 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)

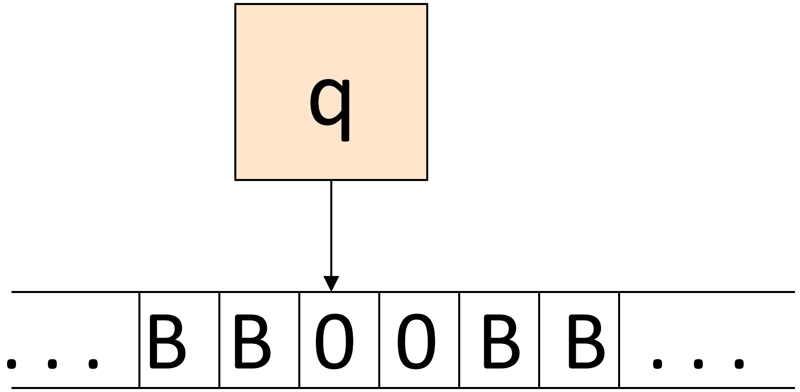
- Состояния = {q (start), f (final)}.
- Входные символы = {0, 1}.
- Символы ленты = {0, 1, B}.
- $\delta(q, 0) = (q, 0, R)$.
- $\delta(q, 1) = (f, 0, R)$.
- $\delta(q, B) = (q, 1, L)$.

Моделирование МТ

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

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

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

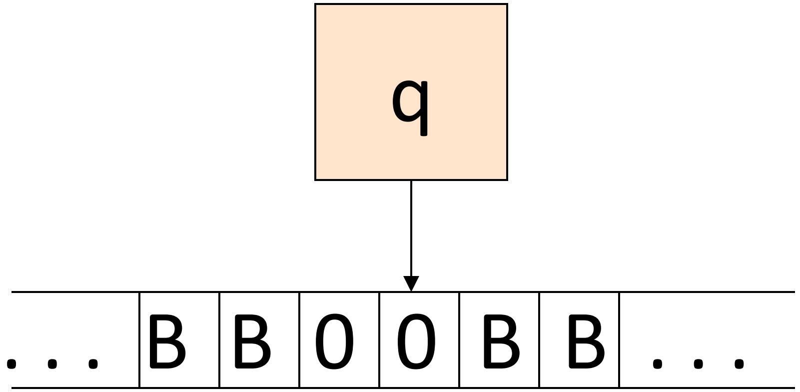


Моделирование МТ

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

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

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

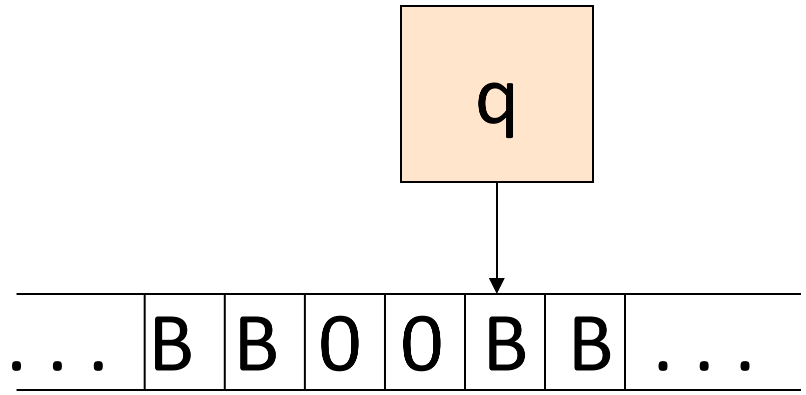


Моделирование МТ

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

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

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

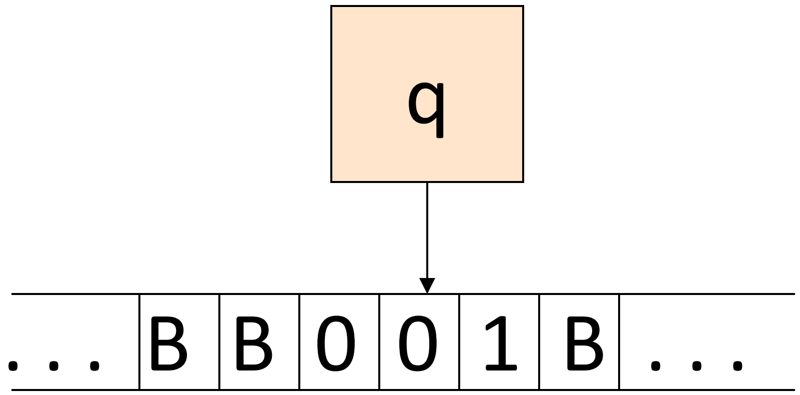


Моделирование МТ

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

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

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

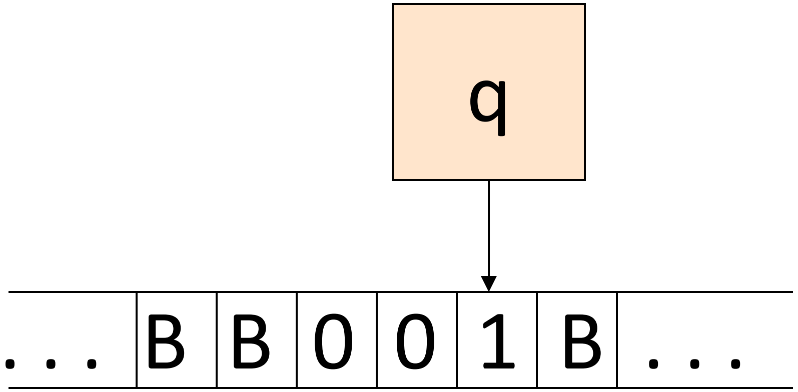


Моделирование МТ

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

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

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

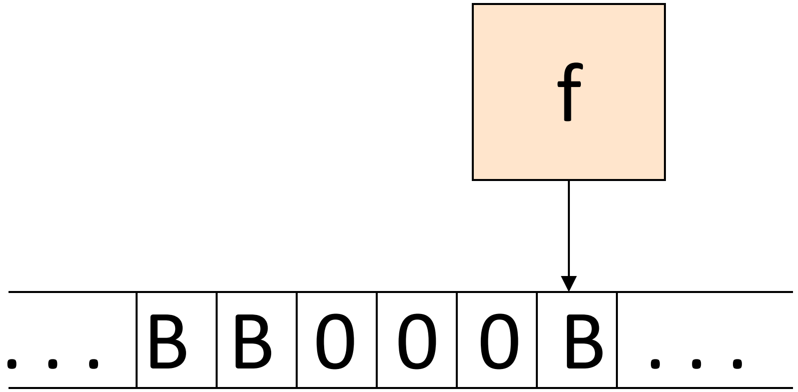


Моделирование МТ

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

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

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



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 читающая головка обозревает 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 с левой стороны ленты, то следующие пробелы до конца справа от q являются частью α.

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
    - $\alpha qZ\beta \vdash \alpha Yp\beta$
    - Если Z есть пробел B, то также $\alpha q \vdash \alpha Yp$
2. Если $\delta(q, Z) = (p, Y, L)$, то
    - Для любого X, $\alpha XqZ\beta \vdash \alpha pXY\beta$
    - Дополнительно, $qZ\beta \vdash pBY\beta$

Языки МТ

- МТ обычно определяет язык по конечному состоянию.
- $L(M) = \{w \mid q_0w \vdash^* I$, где I есть конфигурация с конечным состоянием$\}$.
- Или, МТ может определять язык по остановке.
- $H(M) = \{w \mid q_0w \vdash^* I$, 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 -> Финальное состояние

- Модифицируем M, чтобы получить M” , следующим образом :
    1. Введём новое состояние f, единственное финальное состояние для M”.
    2. МТ в состоянии 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

- Теорема. Классы языков, определяемые машинами Тьюринга с использованием конечных состояний и останова, являются эквивалентными.
- Этот класс языков называется *рекурсивно-перечислимыми языками*.
- Для данного класса языков 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
- Можно реализовать алгоритм CYK для любого КС-языка на машине Тьюринга.
- Почти все, что мы можем вообразить, - рекурсивный язык.
- Очень трудно придумать язык, который не является рекурсивным, разве что с использованием трюков, таких как диагонализация, которую мы обсуждали ранее.

Не рекурсивно-перечислимые языки

- Пусть L_d - бинарный язык, состоящий из всех строк v таких, что МТ М, код которой есть v, не принимает v в качестве своего входа.
- Не существует МТ, принимающей язык L_d .

Свойства замыкания для рекурсивных и рекурсивно-перечислимых языков

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

Свойства замыкания для рекурсивных и рекурсивно-перечислимых языков

Рекурсивные языки

- замкнуты относительно объединения, конкатенации, операции Клини, обращения, пересечения, обратного гомоморфизма.
- замкнут по операциям вычитания и дополнения.

Рекурсивно-перечислимые языки

- замкнуты относительно объединения, конкатенации, операции Клини, обращения, пересечения, обратного гомоморфизма.
- замкнуты по гомоморфизму.

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.

  - Идея универсальной машины Тьюринга не должна показаться странной, если вспомнить о виртуальной машине Java. JVM принимает код программы Java и входные данные для этой программы и выполняет программу на входе.

Универсальный язык (2)

- Рассмотрим язык L_u, 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.
- УМТ принимает код M и w т.и.т.т., когда M принимает w.

Доказательство L_u – рекурсивно-перечислимый, но не рекурсивный

- Построена ТМ для L_u, который конечно, RE.
- Предположим, что он рекурсивный; т.е., мы могли бы построить УМТ U, которая всегда останавливается.
- Тогда мы могли бы также сконструировать алгоритм для L_d, следующим образом:

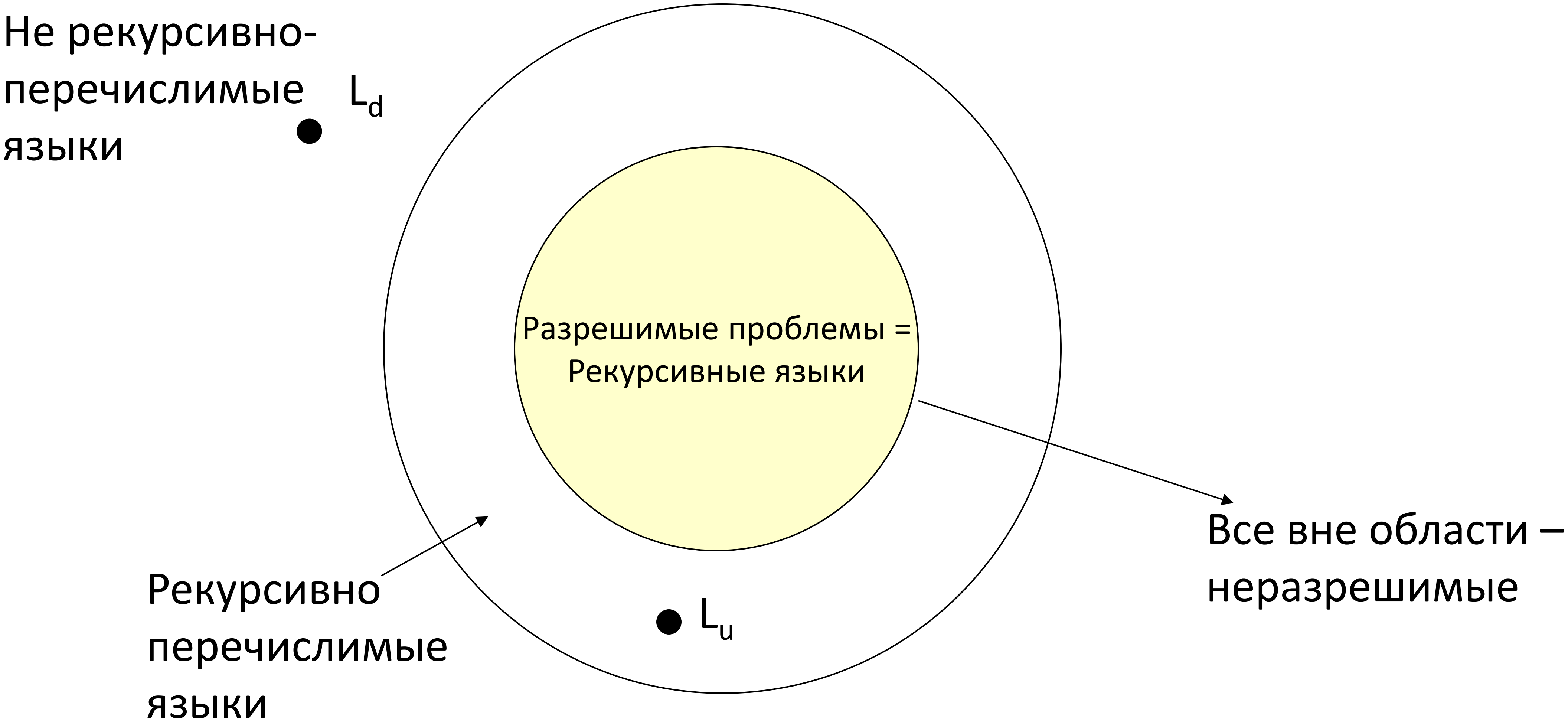
Доказательство – (2)

- Для входа w, мы можем решить, находится ли он в L_d следующим образом:

1. Проверим, что w – правильный код МТ.
   - Если нет, то ее язык пуст, и w в L_d.
2. Если правильный, используем гипотетический алгоритм, для решения, находится ли w111w в L_u.
3. Если это так, то w НЕ в L_d; в противном случае - да.

Доказательство – (3)

- Но мы уже знаем, что не существует алгоритмы для L_d.
- Таким образом, наше предположение о том, что есть алгоритм для L_u неверно.
- L_u – рекурсивно-перечислимый, но не рекурсивный.


Не рекурсивно-перечислимые языки
L_d
Разрешимые проблемы = Рекурсивные языки
Все вне области – неразрешимые
Рекурсивно перечислимые языки
L_u

Проблемы

- Неформально, “проблема” – это вопрос с ответом да/нет о бесконечном множестве возможных *случаев* (исходов).
- Пример: “имеет ли граф G *Гамильтонов цикл* (цикл, который включает точно один раз все вершины)?
  - Каждый ненаправленный граф является примером “Проблемы гамильтонова цикла”.

Проблемы – (2)

- Формально, проблема – это язык.
- Каждая строка кодирует некоторый случай.
- Строка является частью языка, т.и т.т., когда ответом на этот случай проблемы есть «да».

Пример: Проблема о машинах Тьюринга

- Мы можем думать о языке L_d как проблеме.
- “Не принимает ли эта МТ свой собственный код?”

Разрешимые проблемы

- Проблема является *разрешимой* если существует отвечающий за нее алгоритм.
  - Вспомним: “Алгоритм,” формально, есть МТ, которая останавливается на всех входах с принятием или нет последнего.
  - Иными словами, «разрешимая проблема» = «рекурсивный язык».
  - В противном случай, проблема является *неразрешимой*.

От абстракции к реалиям

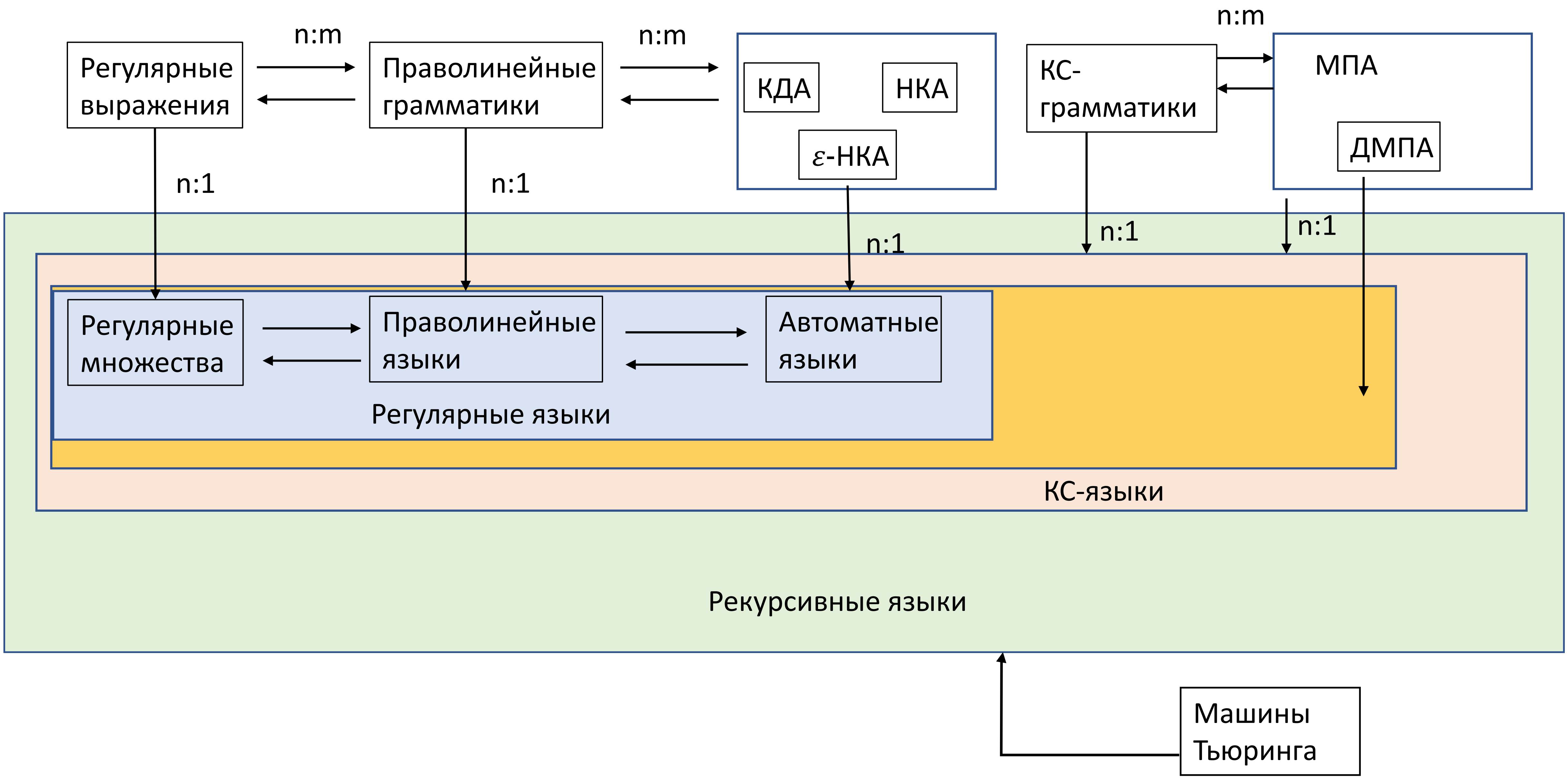
- Хотя факт того, что L_d неразрешим, интересен с интеллектуальной точки зрения, он не влияет на реальный мир напрямую.
- Сначала мы разовьём некоторые имеющие отношение к МТ проблемы, которые неразрешимы, но наша цель состоит в том, чтобы использовать теорию, чтобы показать неразрешимость некоторых реальных проблем.

Пример: Неразрешимые проблемы

- Выполнится ли определенная строка 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 Σ^*?
- Является ли КС-язык регулярным?
- Не существует алгоритма, определяющего для произвольной КС-грамматики, существует ли для нее эквивалентная грамматика, к которой метод рекурсивного спуска применим

Примеры неразрешимых проблем для МТ

- Является ли L(M) регулярным языком?
- Является ли L(M) КС-языком?
- Включает ли L(M) какие-либо палиндромы?
- Пуст ли L(M)?
- Содержит ли L(M) более 1000 строк?
- И.т.д.,.


Регулярные выражения
n:m
Праволинейные грамматики
n:m
КДА
НКА
ε-НКА
КС-грамматики
n:m
МПА
ДМПА
n:1
n:1
n:1
n:1
n:1
Регулярные множества
Праволинейные языки
Автоматные языки
Регулярные языки
КС-языки
Рекурсивные языки
Машины Тьюринга

Синтаксический анализ

Задача синтаксического анализа

На этапе синтаксического анализа нужно:

1. Установить, имеет ли цепочка лексем структуру, заданную синтаксисом языка, и
2. Зафиксировать эту структуру.

- Надо решать задачу разбора: дана цепочка лексем, и надо определить, выводима ли она в грамматике, определяющей синтаксис языка. Если да, то построить *вывод* этой цепочки или *дерево вывода*.
- Для описания синтаксиса языков программирования используют КС-грамматики. Для разных подклассов КС-грамматик построены достаточно эффективные алгоритмы разбора.

Общие алгоритмы синтаксического анализа

- **Существует алгоритм, который по любой данной КС-грамматике и данной цепочке выясняет, принадлежит ли цепочка языку, порождаемому этой грамматикой.**
- Но время работы такого алгоритма (синтаксического анализа с возвратами) экспоненциально зависит от длины цепочки!
- Существуют табличные методы анализа, применимые ко всему классу КС- грамматик и требующие для разбора цепочек длины *n* времени Cn^3 (алгоритм Кока-Янгера- Касами), где С — константа, либо Cn^2 (алгоритм Эрли).
- **Алгоритмы анализа, расходующие на обработку входной цепочки линейное время, применимы только к некоторым подклассам КС-грамматик.**

- Каждый метод предполагает свой способ построения по грамматике программы-анализатора, которая будет осуществлять разбор цепочек.
- **Корректный анализатор** завершает свою работу для любой входной цепочки и выдает верный ответ о принадлежности цепочки языку.
- Анализатор не корректен, если:
  - не распознает хотя бы одну цепочку, принадлежащую языку;
  - распознает хотя бы одну цепочку, языку не принадлежащую;
  - зацикливается на какой-либо цепочке.

- Говорят, что **метод анализа применим к данной грамматике, если анализатор, построенный в соответствии с этим методом, корректен**.

Метод рекурсивного спуска

- Пример
- $G_1 = \langle \{a, b, c, d\}, \{S, A, B\}, P, S \rangle$, где P:

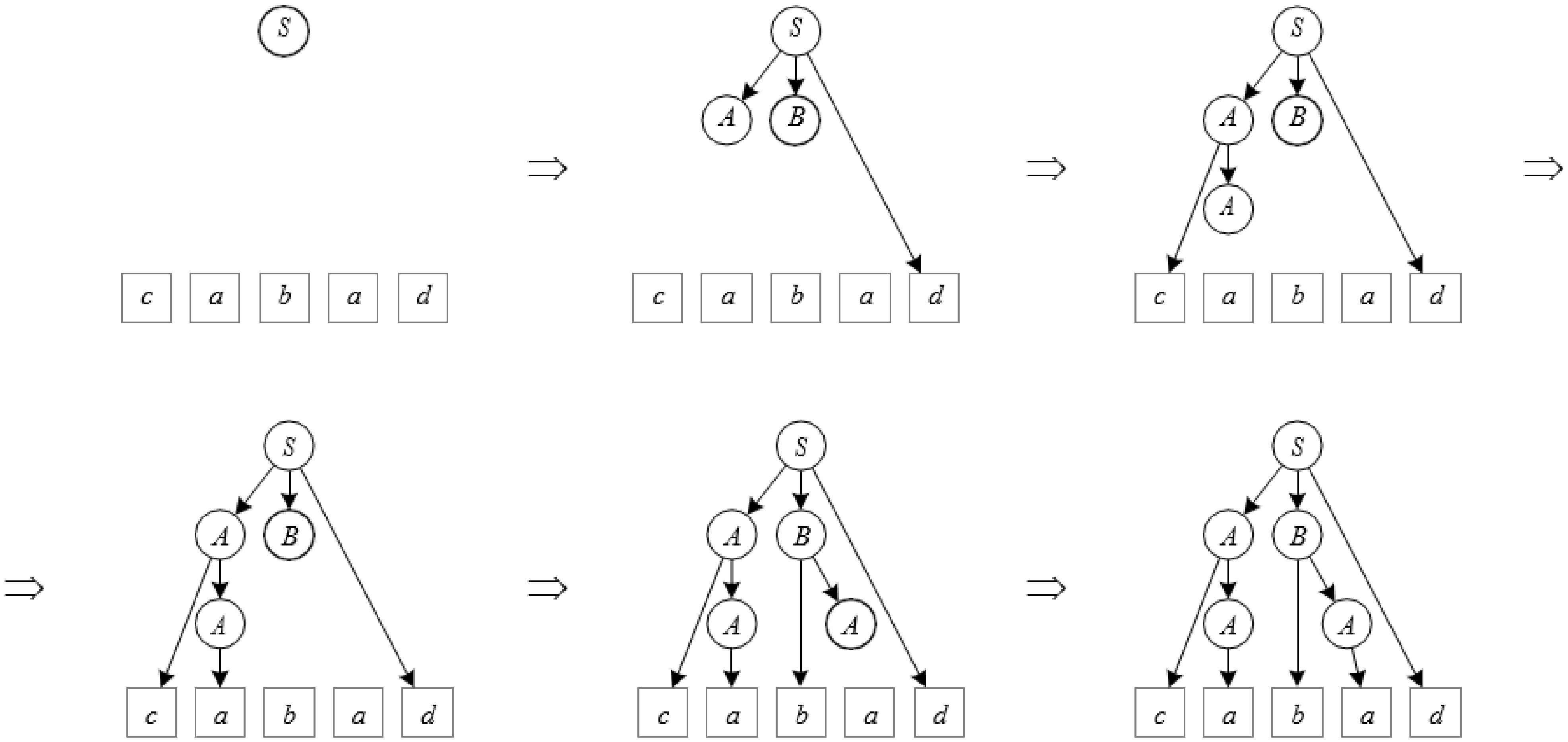
$$
\begin{aligned}
S &\to ABd \\
A &\to a \mid cA \\
B &\to bA
\end{aligned}
$$

- Принадлежит ли цепочка *cabad* языку $L(G_1)$?
- Построим левый вывод этой цепочки:

$$S \to ABd \to cABd \to caBd \to cabAd \to cabad.$$

- Следовательно, цепочка *cabad* принадлежит языку $L(G_1)$.

Дерево нисходящего разбора



Метод рекурсивного спуска

- Метод рекурсивного спуска (РС-метод) реализует разбор сверху-вниз и делает это с помощью системы рекурсивных процедур.
- Для каждого нетерминала грамматики создается своя процедура, носящая его имя; ее задача — начиная с указанного места исходной цепочки найти подцепочку, которая выводится из этого нетерминала.
  - Если такую подцепочку найти не удается, то процедура завершает свою работу, сигнализируя об ошибке. Это означает, что цепочка не принадлежит языку; разбор останавливается.
  - Если подцепочку удалось найти, то работа процедуры считается нормально завершенной и осуществляется возврат в точку вызова.

Метод рекурсивного спуска

- Каждая процедура определяется по правилам вывода (по альтернативам) соответствующего нетерминала:
  - для правой части каждого правила осуществляется поиск подцепочки, выводимой из этой правой части. При этом терминалы из правой части распознаются самой процедурой, а нетерминалы соответствуют вызовам процедур, носящих их имена.
- После распознавания каждого терминала процедура считывает следующий символ из исходной цепочки, который становится текущим анализируемым символом. Выбор нужной альтернативы (правила) осуществляется процедурой по первому символу из еще нерассмотренной части исходной цепочки (т. е. по текущему символу).
- **Выбор нужной альтернативы при анализе методом рекурсивного спуска легко осуществим, если все альтернативы начинаются с попарно различных терминальных символов.**
- Метод рекурсивного спуска является одной из возможных реализаций нисходящего анализа с прогнозируемым выбором альтернатив.

- Выбор нужной альтернативы на очередном шаге вывода в грамматике G1 представим в виде таблицы прогнозов

P:

$$
\begin{aligned}
S &\rightarrow ABd \\
A &\rightarrow a \mid cA \\
B &\rightarrow bA
\end{aligned}
$$

	a	**b**	**c**	**d**
S	S → ABd	S → ABd	S → ABd	S → ABd
A	A → a		A → cA	
B		B → bA		

P:

$$
\begin{aligned}
S &\rightarrow ABd \\
A &\rightarrow a \mid cA \\
B &\rightarrow bA
\end{aligned}
$$

- w = *cabad*

S() : S ->ABd

A() : A ->cA

- w = _*abad*

A() : A ->a

- w = _ _*bad*

B() : *B* ->bA

w = _ _ _*ad*

A() : A ->a

- w = _ _ _ _***d***
- *Разбор закончен*

	a	**b**	**c**	**d**
S	S → ABd	S → ABd	S → ABd	S → ABd
A	A → a		A → cA	
B		B → bA		

Достаточное условие применимости метода рекурсивного спуска

- Для применимости метода рекурсивного спуска достаточно, чтобы каждое правило в грамматике удовлетворяло одному из двух видов:

  (а) $X \to \alpha$, где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для этого нетерминала X;

  (б) $X \to a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$, где $a_i \in T$ для всех $i = 1, 2, \ldots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, т. е. если для нетерминала X есть несколько правил вывода, то они должны начинаться с терминалов, причем все эти терминалы должны быть попарно различными.
- Это условие не является необходимым.
- КС - грамматику, удовлетворяющую данному условию, называют ***s-грамматикой*.**

Применимость метода РС

- Метод рекурсивного спуска применим к грамматике, если для нее существует таблица однозначных прогнозов.
- Не для каждой КС-грамматики существует таблица с однозначными прогнозами, позволяющая безошибочно осуществить выбор альтернативы на каждом шаге вывода.
- Таким образом, нисходящий анализ с прогнозируемым выбором альтернатив пригоден лишь для некоторого подкласса КС-грамматик.
- Метод рекурсивного спуска (без возвратов) неприменим к неоднозначным грамматикам:

$$G_2:$$

$$\begin{aligned} S &\to aA \mid B \mid d \\ A &\to d \mid aA \\ B &\to aA \mid a \end{aligned}$$

Применимость метода РС

- Грамматика может быть однозначной, однако однозначных прогнозов для нее может не быть:

$$G_3:$$

$$\begin{aligned} S &\to A \mid B \\ A &\to aA \mid d \\ B &\to aB \mid b \end{aligned}$$

- Каждая цепочка, выводимая в G_3 из S, оканчивается либо символом b, либо символом d, и имеет единственное дерево вывода.
- Но невозможно предсказать, с какой альтернативы ($S \to A$ или $S \to B$) начинать вывод, не просмотрев всю цепочку до конца и не увидев последний символ.

Применимость метода РС

<u>Опр.</u> Множество *first* (α) в грамматике $G = \langle\, T, N, P, S \,\rangle$ — это множество терминальных символов, которыми начинаются цепочки, выводимые в G из цепочки α:

$\alpha \in (\, T \cup N \,)^*$, т. е. $first\,(\alpha) = \{\, a \in T \mid \alpha \Rightarrow a\alpha',\ \ \alpha' \in (\, T \cup N \,)^* \,\}$.

- Например, для альтернатив правила $S \rightarrow A \mid B$ в грамматике G_3 имеем: $first\,(A) = \{\, a, d \,\}$, $first\,(B) = \{\, a, b \,\}$.
- Пересечение этих множеств непусто: $first\,(A) \cap first\,(B) = \{\, a \,\} \neq \varnothing$, и поэтому метод рекурсивного спуска к G_3 неприменим.
- **Наличие в грамматике правила с альтернативами $X \rightarrow \alpha \mid \beta$, такими что $first\,(\,\alpha\,) \cap first\,(\,\beta\,) \neq \varnothing$, делает метод рекурсивного спуска неприменимым.**

Рассмотрим еще несколько примеров.

G_4:

$$S \to aA \mid BDc$$
$$A \to BAa \mid aB \mid b$$
$$B \to \varepsilon$$
$$D \to B \mid b$$

$first(aA) = \{ a \}, \quad first(BDc) = \{ b, c \};$

$first(BAa) = \{ a, b \}, \quad first(aB) = \{ a \},$

$first(b) = \{ b \};$

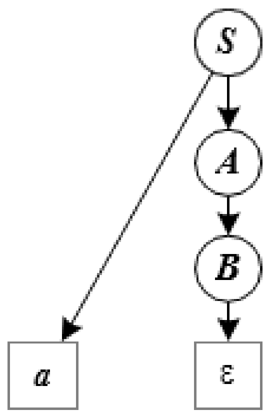
$first(\varepsilon) = \varnothing;$

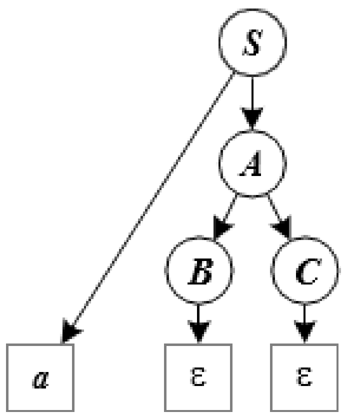
$first(B) = \varnothing, \quad first(b) = \{ b \}.$

Метод рекурсивного спуска неприменим к грамматике G_4, так как $first\,(BAa) \cap first\,(aB) = \{ a \} \neq \varnothing$.

G_5:

$$
\begin{aligned}
S &\to aA \\
A &\to BC \mid B \\
C &\to b \mid \varepsilon \\
B &\to \varepsilon
\end{aligned}
$$





- Пересечение множеств *first* пусто для любой пары альтернатив грамматики G_5, однако наличие двух различных альтернатив, из которых выводится пустая цепочка, делает данную грамматику неоднозначной и, следовательно, метод рекурсивного спуска к ней неприменим.
- Действительно, $BC \Rightarrow \varepsilon$ и $B \Rightarrow \varepsilon$. Цепочка а имеет два различных дерева вывода
- Если в грамматике для правила $\boldsymbol{X} \to \alpha \mid \beta$ выполняются соотношения $\alpha \Rightarrow \varepsilon$ и $\beta \Rightarrow \varepsilon$**,** то метод рекурсивного спуска неприменим.
- Для грамматики, имеющей для каждого нетерминала не более одной альтернативы, из которой выводится пустая цепочка, метод рекурсивного спуска применим

Рассмотрим примеры с единственной альтернативой, из которой выводится ε.

G_6:

$S \to cAd \mid d$

$A \to aA \mid \varepsilon$

Метод применим: если текущий символ a, то выбираем альтернативу $A \to aA$ иначе — $A \to \varepsilon$

G_7:

$S \to Bd$

$B \to c\underline{Aa} \mid a$

$A \to aA \mid \varepsilon$

Неприменим, т.к. для A невозможно правильно выбрать альтернативу без «заглядывания» на символ вперед.

Опр. Множество $follow(A)$ — это множество терминальных символов, которые могут появляться в сентенциальных формах грамматики $G = \langle T, N, P, S \rangle$ непосредственно справа от A (или от цепочек, выводимых из A), т.е.

$$follow(A) = \{ a \in T \mid S \Rightarrow \alpha A \beta,\ \beta \Rightarrow a\gamma,\ A \in N,\ \alpha, \beta, \gamma \in (T \cup N)^* \}.$$

- Если в грамматике есть пара правил $X \to \alpha \mid \beta$, таких, что $\beta \Rightarrow \varepsilon$, $first(\alpha) \cap follow(X) \neq \varnothing$, то метод рекурсивного спуска неприменим к данной грамматике.

Критерий применимости метода рекурсивного спуска

Теорема. Пусть G — КС-грамматика. Метод рекурсивного спуска применим к G, т. и т. т., когда для любой пары альтернатив

$X \to \alpha \mid \beta$ выполняются следующие условия:

1) $first(\alpha) \cap first\,(\beta) = \varnothing$;
2) справедливо не более чем одно из двух соотношений:

$$\alpha \Rightarrow \varepsilon,\ \beta \Rightarrow \varepsilon\ ;$$

3) если $\beta \Rightarrow \varepsilon$, то $first(X) \cap follow(X) = \varnothing$.

- Не существует алгоритма, определяющего для произвольной КС-грамматики, существует ли для нее эквивалентная грамматика, к которой метод рекурсивного спуска применим (т. е. это алгоритмически неразрешимая проблема).

Преобразования грамматик

- Если грамматика не удовлетворяет требованиям применимости метода рекурсивного спуска, то можно попытаться преобразовать ее, т. е. получить эквивалентную грамматику, пригодную для анализа этим методом.
- Каждый КС-язык определяется нелеворекурсивной грамматикой (НФГ).
- **Опр**. КС-грамматика $G = (N, \Sigma, P, S)$ называется *грамматикой в нормальной форме Грейбах* , если в ней нет ε -правил и каждое правило из *P*, отличное от $S \to \varepsilon$, имеет вид $A \to a\alpha$, где $a \in \Sigma$ и $\alpha \in N^*$.

Нормальная форма Грейбах

Нормальная форма Грейбах

- Для каждого КС-языка можно найти грамматику, все правые части правил которых начинаются с терминалов.
- **<u>Опр</u>**. Нетерминал A грамматики $G=(N,\Sigma,P,S)$ называется *рекурсивным*, если $A\Rightarrow^*\alpha A\beta$ для некоторых α и β. 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, рекурсивые, 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:

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

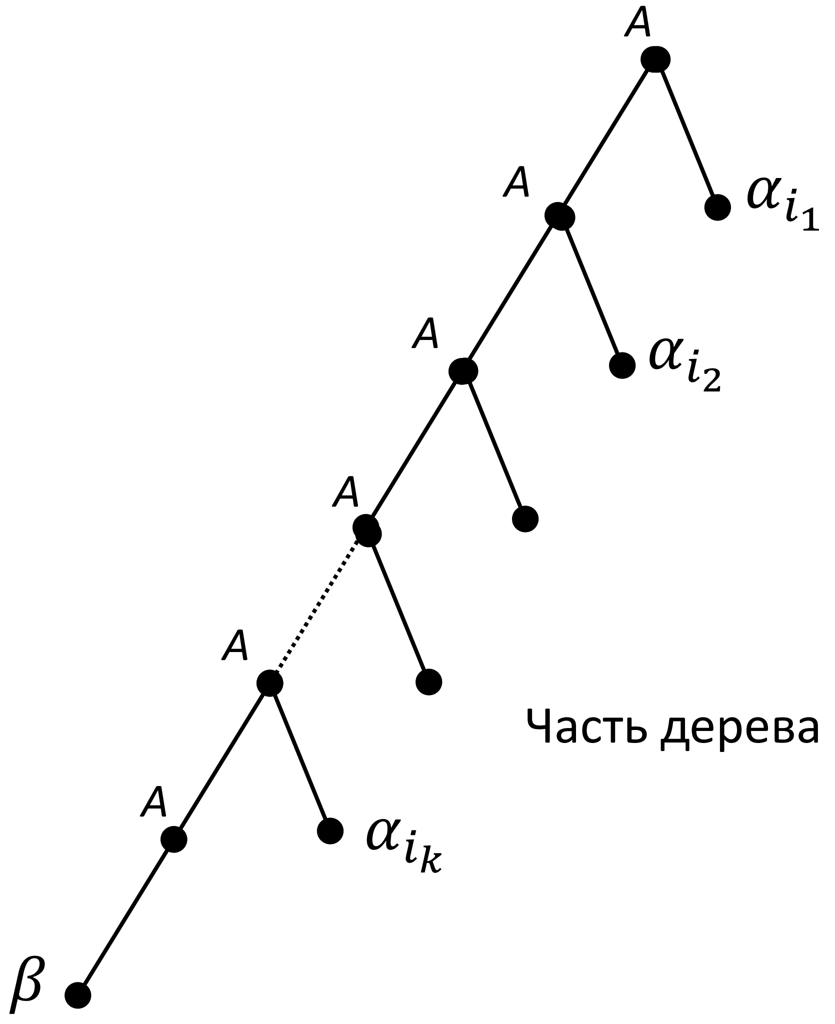
и ни одна цепочка из β_i не начинается с A.

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

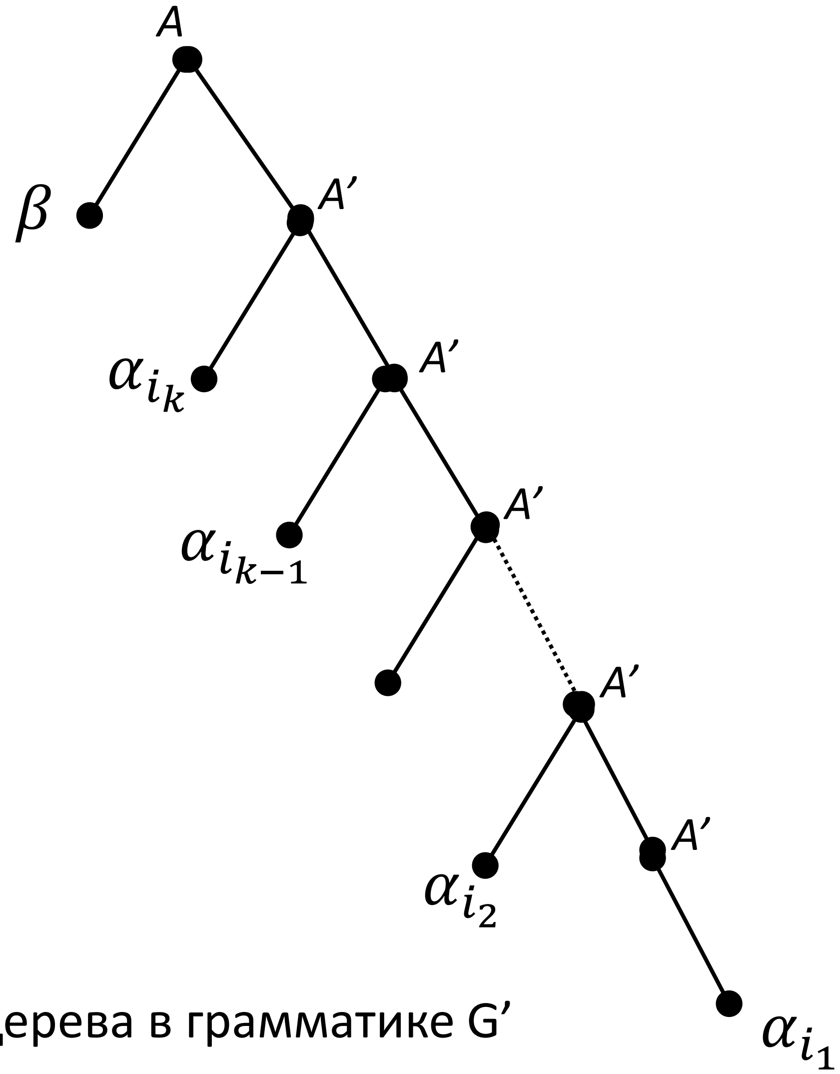
$$A \to \beta_1 | \beta_2 | \dots | \beta_n | \beta_1 A' | \beta_2 A' | \dots | \beta_n A'$$
$$A' \to \alpha_1 | \alpha_2 | \dots | \alpha_m | \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A'.$$

Тогда $L(G') = L(G)$.

Иллюстрация преобразования правил



Часть дерева в грамматике G



Часть дерева в грамматике G′

Доказательство.

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

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

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

Все шаги вывода в G, в которых не используются А-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. $\left.\begin{array}{l} L(G') \subseteq L(G) \\ L(G) \subseteq L(G') \end{array}\right. \Longrightarrow L(G') = L(G)$

Алгоритм. ” Устранение левой рекурсии”

1. Пусть N = $\{A_1, \ldots, A_n\}$. Преобразуем G таким образом, чтобы в правиле $A_i \rightarrow \alpha$ цепочка α начиналась либо с терминала, либо с такого A_j , что $j > i$.

Положим $i = 1$

2. Пусть множество A_i-правил — это $A_i \rightarrow A_i\alpha_1 \mid \ldots \mid A_i\alpha_m \mid \beta_1 \mid \ldots \mid \beta_p$, где ни одна из цепочек β_j не начинается с A_k, если $k \leq i$.

Заменим A_i-правила правилами:

$A \rightarrow \beta_1 \mid \ldots \mid \beta_p \mid \beta_1 A'i \mid \ldots \mid \beta_p A'i$

$A'_i \rightarrow \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 A'_i \mid \ldots \mid \alpha_m A'_i$

где A'_i — новый нетерминал.

Правые части всех A_i-правил начинаются теперь с терминала или с A_k для некоторого $k > j$.

Алгоритм. ” Устранение левой рекурсии”

3. Если i = n, полученную грамматику G’ считать результатом и остановиться.

В противном случае положить i = i + 1 и j = 1.

4. Заменить каждое правило вида $A_i \to A_j\alpha$ правилами

$$A_i \to \beta_1\alpha \mid \ldots \mid \beta_m\alpha,$$

где $A_j \to \beta_1 \mid \beta_m$ — все A_j -правила.

Так как правая часть каждого Aj -правила начинается уже с терминала или с A_k для k > j, то и правая часть каждого A_i-правила будет теперь обладать этим свойством.

5. Если j = i − 1, нужно перейти к шагу (2). В противном случае, положить j = j + 1 и перейти к шагу (4).

Устранение непосредственной левой рекурсии

- S→Aβ
- A→Sα|Aα

Есть непосредственная левая рекурсия A→Aα

Добавим нетерминал A′ и добавим правила A→SαA′ , A′→αA′, A′→α

$$A \to A\alpha_1|A\alpha_2|\ldots|A\alpha_m|\beta_1|\beta_2|\ldots|\beta_n$$
$$\downarrow$$
$$A \to \beta_1|\beta_2|\ldots|\beta_n|\beta_1A'|\beta_2A'|\ldots|\beta_nA'$$
$$A' \to \alpha_1|\alpha_2|\ldots|\alpha_m|\alpha_1A'|\alpha_2A'|\ldots|\alpha_mA'.$$

Новая грамматика:

- S→Aβ
- A→SαA′|Sα
- A′→αA′|α

В новой грамматике нет непосредственной левой рекурсии, но нетерминал A леворекурсивен, так как есть A⇒SαA′⇒AβαA′

Пример

Задача. Устраните левую рекурсию в следующей грамматике:

S → AA | 0

A → SS | 1

Решение. Составим множество N таким образом:

N = {S, A}

Выполняя описанный алгоритм (шаг 4), сначала получим:

S → AA | 0

A → AAS | 0S | 1

В итоге (после применения шага 2) грамматика примет вид:

S → AA | 0

A → 0S | 1 | 0SA’| 1A’

A’ → AS | ASA’

Пример

- Дана грамматика:

  $A \to S\alpha$

  $S \to S\beta \mid A\gamma \mid \beta$

- Среди правил A непосредственной рекурсии нет, поэтому во время первой итерации внешнего цикла ничего не происходит. Во время второй итерации внешнего цикла правило $S \to A\gamma$ переходит в $S \to S\alpha\gamma$

- Грамматика примет вид:

  $A \to S\alpha$

  $S \to S\beta \mid S\alpha\gamma \mid \beta$

- Устраняем левую рекурсию для S

  $S \to \beta S_1 \mid \beta$

  $S_1 \to \beta S_1 \mid \alpha\gamma S_1 \mid \beta \mid \alpha\gamma$

Пример

- G:

E->E+T|T

T->T*F|F

F->(E)|a

- G’:

E->T|TE’

E’->+T|+TE’

T->F|FT’

T’->*F|*FT’

F->(E)|a

Пример

Исходная грамматика

уже в НФХ

$S \to XA | BB$

$B \to b | SB$

$X \to b$

$A \to a$

Упорядочим переменные

- $S < X < A < B$

Удаление стартового нетерминала из правых частей правил

$S \to XA | BB$

$B \to bAB | BBB | b$

$X \to b$

$A \to a$

Удаление левой рекурсии

$S \to XA | BB$

$B \to bAB | b | bABZ | bZ$

$Z \to BB | BBZ$

$X \to b$

$A \to a$

Обработка правила

$S \to XA|BB$

$S \to bA | bABB | bB | bABZB | bZB$

$B \to bAB | b | bABZ | bZ$

$Z \to BB | BBZ$

$X \to b$

$A \to a$

Пример (прод.)

Обработка правила $Z \to BB|BBZ$

S→bA|bABB|bB|bABZB|bZB

B→bAB|b|bABZ|bZ

Z→bABB|bB|bABZB|bZB|bABBZ|bBZ|bABZBZ|bZBZ

X→b

A→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 ε -правил и каждое правило из P, отличное от $S \to \varepsilon$, имеет вид $A \to a\alpha$, где $a \in \Sigma$ и $\alpha \in N^*$.

- **<u>Лемма</u>**. Пусть - не леворекурсивная грамматика. Существует такой порядок < на N, что если $A \rightarrow B\alpha$ принадлежит P, то $A < B$.

- **<u>Теорема</u>**. Если L-КС-язык, то L=L(G) для некоторой грамматики G в нормальной форме Грейбах.

Применение

- **Простота доказательств**
- Использование нормальных форм существенно упрощает доказательство теорем. Например, использование нормальной формы Грейбах позволяет доказать, что для каждого контекстно-свободного языка (не содержащего ε) существует автомат с магазинной памятью без переходов по ε

- **Разбор грамматики**
- Нормальная форма Хомского позволяет производить разбор грамматики.
- В свою очередь, нормальная форма Грейбах позволяет использовать метод рекурсивного спуска, сложность которого является линейной, несмотря на возвраты.

Преобразования грамматик

- Если в грамматике есть нетерминал, у которого несколько правил вывода начинаются **одинаковыми терминальными символами**, т. е. имеют вид

$$A \to a\alpha_1 \mid a\alpha_2 \mid \dots \mid a\alpha_n \mid \beta_1 \mid \dots \mid \beta_m,$$

- где $a \in T$; α_i, $\beta_j \in (T \cup N)^*$, β_j не начинается с a, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$, то непосредственно применять метод рекурсивного спуска нельзя, т. к. $first(a\alpha_i) \cap first(a\alpha_k) \neq \varnothing$ для $i \neq k$.
- Можно преобразовать правила вывода данного нетерминала, объединив правила с общими началами в одно правило:

$$A \to aA' \mid \beta_1 \mid \dots \mid \beta_m$$
$$A' \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

- Будет получена грамматика, эквивалентная данной.

Преобразования грамматик

- Если в грамматике есть нетерминал, у которого **несколько** правил вывода, и среди них есть правила, **начинающиеся нетерминальными символами**, т. е. имеют вид:

$$
\begin{aligned}
A &\to B_1\alpha_1 \mid \dots \mid B_n\alpha_n \mid a_1\beta_1 \mid \dots \mid a_m\beta_m \\
B_1 &\to \gamma_{11} \mid \dots \mid \gamma_{1k} \\
&\dots \\
B_n &\to \gamma_{n1} \mid \dots \mid \gamma_{np},
\end{aligned}
$$

где $B_i \in N$; $a_j \in T$; α_i, β_j, $\gamma_{ij} \in (T \cup N)^*$, то можно заменить вхождения нетерминалов B_i их правилами вывода в надежде, что правила нетерминала A станут удовлетворять условиям применимости метода рекурсивного спуска:

$$A \to \gamma_{11}\alpha_1 \mid \dots \mid \gamma_{1k}\alpha_1 \mid \dots \mid \gamma_{n1}\alpha_n \mid \dots \mid \gamma_{np}\alpha_n \mid a_1\beta_1 \mid \dots \mid a_m\beta_m$$

Преобразования грамматик

- Если есть правила с пустой альтернативой вида:

$A \to \alpha_1 A \mid \ldots \mid \alpha_n A \mid \beta_1 \mid \ldots \mid \beta_m \mid \varepsilon$

$B \to \alpha A \beta$

и $first(A) \cap follow(A) \neq \varnothing$ (из-за вхождения A в правила вывода для B), то можно построить такую грамматику:

$B \to \alpha A'$

$A' \to \alpha_1 A' \mid \ldots \mid \alpha_n A' \mid \beta_1\beta \mid \ldots \mid \beta_m\beta \mid \beta$

- Полученная грамматика будет эквивалентна исходной, т.к. из B по-прежнему выводятся цепочки вида $\alpha\ \{\alpha_i\}\ \beta_j\ \beta$ либо $\alpha\ \{\alpha_i\}\ \beta$.
- Однако правило вывода для нетерминального символа A' будет иметь альтернативы, начинающиеся одинаковыми терминальными символами (т. к. $first(A) \cap follow(A) \neq \varnothing$) ; следовательно, потребуются дальнейшие преобразования, и успех не гарантирован.

Пример

- Рассмотрим грамматику G_{origin}:

$$
\begin{array}{l}
G_{origin} \\
\quad S \rightarrow fASd \mid \varepsilon \\
\quad A \rightarrow Aa \mid Ab \mid dB \mid f \\
\quad B \rightarrow bcB \mid \varepsilon
\end{array}
$$

$$first(Aa) = first(Ab) = \{d, f\}$$
$$first(bcB) = \{b\},\ follow(B) = \{a, b, d, f\}$$

- Условия применимости метода рекурсивного спуска не выполняются для G_{origin}.
- С помощью преобразований приведем эту грамматику к каноническому виду для рекурсивного спуска.

G_{origin}:

$$
\begin{aligned}
S &\rightarrow fASd \mid \varepsilon \\
A &\rightarrow Aa \mid Ab \mid dB \mid f \\
B &\rightarrow bcB \mid \varepsilon
\end{aligned}
$$

$\Rightarrow_{(1)}$

Удаление левой рекурсии

$G_{transform_1}$:

$$
\begin{aligned}
S &\rightarrow fASd \mid \varepsilon \\
A &\rightarrow dBA' \mid fA' \\
A' &\rightarrow aA' \mid bA' \mid \varepsilon \\
B &\rightarrow bcB \mid \varepsilon
\end{aligned}
$$

$\Rightarrow_{(4)}$

Удаление пустой альтернативы

$$first(S) = \{f\},\ follow(S) = \{d\},\quad first(S) \cap follow(S) = \varnothing;$$

$$first(A') = \{a, b\},\ follow(A') = \{f, d\},\ first(A') \cap follow(A') = \varnothing;$$

$$first(B) = \{b\},\ follow(B) = \{a, b, f, d\},\ first(B) \cap follow(B) = \{b\} \neq \varnothing.$$

$G_{transform_2}$:

$$
\begin{aligned}
S &\rightarrow fASd \mid \varepsilon \\
A &\rightarrow dB' \mid fA' \\
B' &\rightarrow bcB' \mid A' \\
A' &\rightarrow aA' \mid bA' \mid \varepsilon
\end{aligned}
$$

$\Rightarrow_{(4)}$

Удаление пустой альтернативы

$\Rightarrow_{(3)}$

Обработка правил, начинающиеся нетерминальными символами

$G_{transform_3}$:

$$
\begin{aligned}
S &\rightarrow fASd \mid \varepsilon \\
A &\rightarrow dB' \mid fA' \\
B' &\rightarrow bcB' \mid aA' \mid bA' \mid \varepsilon \\
A' &\rightarrow aA' \mid bA' \mid \varepsilon
\end{aligned}
$$

$G_{transform4}$:

$$\begin{aligned} S &\to fASd \mid \varepsilon \\ A &\to dB' \mid fA' \\ B' &\to bC \mid aA' \mid \varepsilon \\ C &\to cB' \mid A' \\ A' &\to aA' \mid bA' \mid \varepsilon \end{aligned}$$

$\Rightarrow_{(2)}$

Обработка правил , начинающимися одинаковыми терминальными символами

Обработка правил, начинающиеся нетерминальными символами

$\Rightarrow_{(3)}$

G_{object}:

$$\begin{aligned} S &\to fASd \mid \varepsilon \\ A &\to dB' \mid fA' \\ B' &\to bC \mid aA' \mid \varepsilon \\ C &\to cB' \mid aA' \mid bA' \mid \varepsilon \\ A' &\to aA' \mid bA' \mid \varepsilon \end{aligned}$$

$$first(B') = \{a, b\}, follow(B') = \{f, d\}; first(B') \cap follow(B') = \varnothing;$$

$$first(A') = \{a, b\}, follow(A') = \{f, d\}; first(A') \cap follow(A') = \varnothing;$$

$$first(C) = \{a, b, c\}, follow(C) = \{f, d\}; first(C) \cap follow(C) = \varnothing.$$

- Т. е. получили эквивалентную грамматику G_{object}, к которой применим метод рекурсивного спуска.
- G_{object} удобна для построения системы рекурсивных процедур, так как ее правила имеют канонический вид.