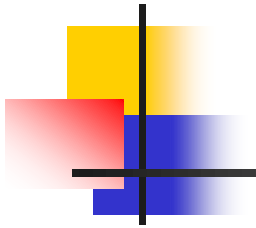
Автоматы и формальные языки

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского Политехнического университета

karpov@dcn.infos.ru



Объявление

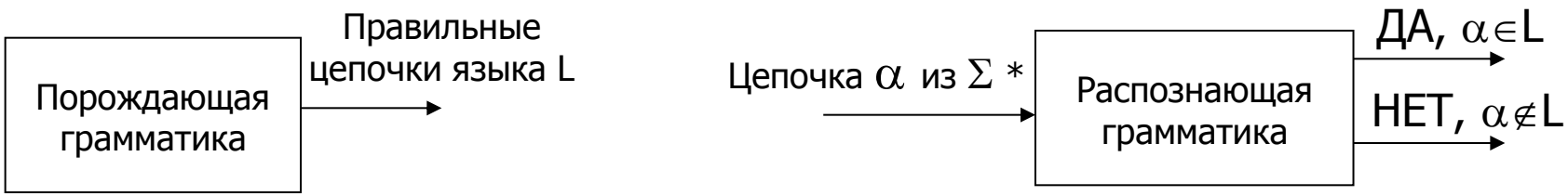
Начиная с 5 марта
по средам с 17 часов в к.106 корп. 9
студент 6 курса
Руслан Морозов
проводит консультации по решению задач и выполнению курсовых работ

Структура курса

- Конечные автоматы-распознаватели – 4 л
- Порождающие грамматики Хомского – 3 л
  - Лекция 5. Синтаксически-ориентированная трансляция и грамматики Хомского
  - Лекция 6. Иерархия грамматик Хомского
  - Лекция 7. Абстрактные распознающие автоматы
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции - 2 л

Порождающая и распознающая грамматики

Грамматика – конечная модель задания языка



- Существует два типа грамматик: порождающие и распознающие
  - *Порождающая грамматика* языка L - это конечный набор правил, позволяющих строить все "*правильные*" предложения языка L, и применение которых не дает ни одного "неправильного" предложения, не принадлежащего L
  - *Распознающая грамматика* задает критерий принадлежности произвольной цепочки данному языку. Это, фактически, некоторый алгоритм, принимающий в качестве входа символ за символом произвольную цепочку над словарем V и дающий на выходе один из двух возможных ответов: "*данная цепочка принадлежит языку L*" либо "*данная цепочка **не** принадлежит языку L*"

Порождающая грамматика языка Милан (Mini Language)

```
begin
    m:=read;
    n:=read;
    while m≠n do
      if m>n then m:=m-n else n:=n-m;
    write(m)
end
```

T	Правила	Название
2	A → γ	Контекстно-свободные грамматики (КС-грамматики)

<программа> → ***begin*** <список операторов> ***end***
<список операторов> → <оператор>**;** <список операторов>
<список операторов> → <оператор>
<оператор> → ***if*** <условие> ***then*** <список операторов> ***else*** <список операторов>
<оператор> → ***if*** <условие> ***then*** <список операторов>
<оператор> → ***while*** <условие> ***do*** <список операторов>
<оператор> → ***i*** := <выражение>
<оператор> → ***write*** (<выражение>)
<условие> → <выражение> ***q*** <выражение>
<выражение> → **(**<выражение>**)** | ***read*** | ***i*** | **c**
| <выражение> **+** <выражение>
| <выражение> **-** <выражение>
| <выражение> ***** <выражение>
| <выражение> **/** <выражение>

Вывод программы на языке Милан

```
begin
    m:=read;
    n:=read;
    while m≠n do
      if m>n then m:=m-n
      else n:=n-m;
    write(m)
end
```

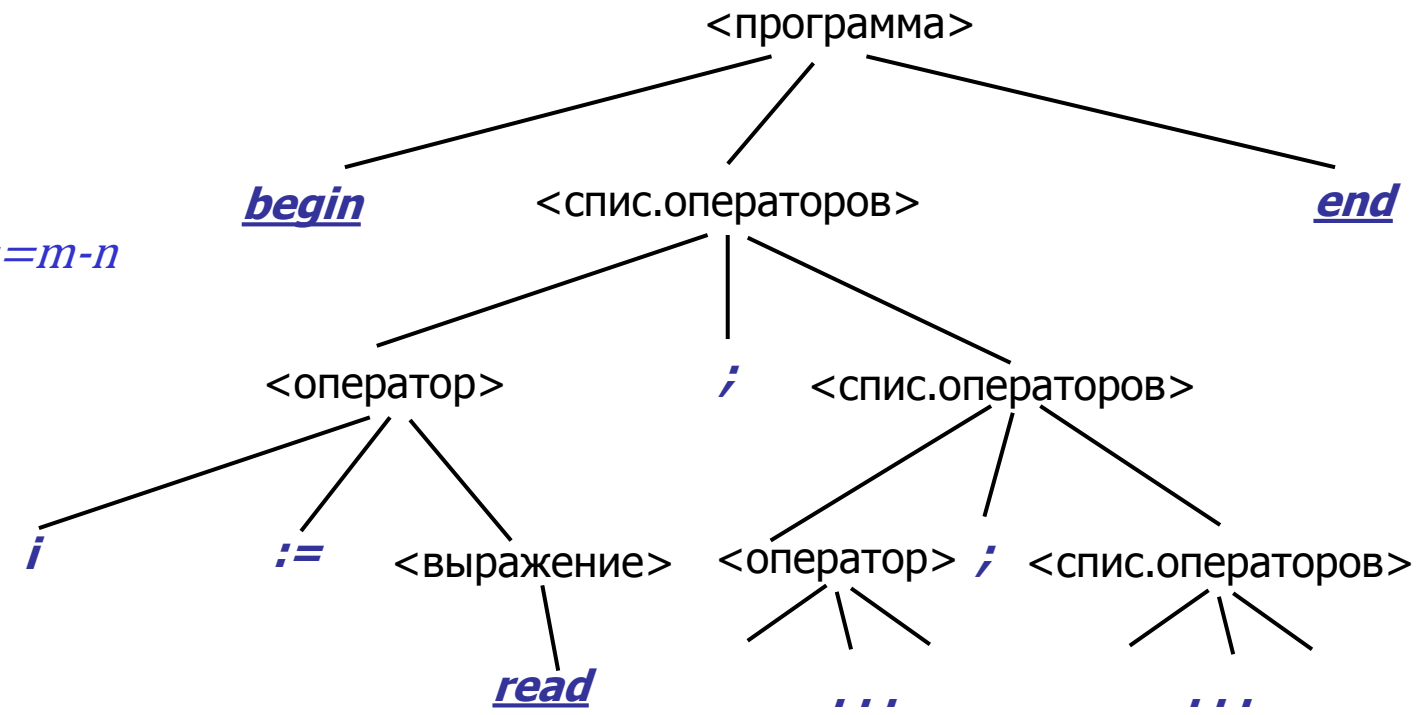
<программа> → **begin** <спис. операторов> **end**
<спис.операторов> → <оператор>**;** <спис. операторов>
<спис.операторов> → <оператор>
<оператор> → **if** <условие> **then** <спис.операторов>
else <спис.операторов>
<оператор> → **if** <условие> **then** <спис.операторов>
<оператор> → **while** <условие> **do** <спис.операторов>
<оператор> → ***i*** := <выражение>
<оператор> → **write** (<выражение>)
<условие> → <выражение> ***q*** <выражение>
<выражение> → **(**<выражение>**)** | **read** | ***i*** | **c** | **...**

Вывод

<программа>
⇒ **begin** <спис. операторов> **end**
⇒ **begin** <оператор>**;** <спис. операторов> **end**
⇒ **begin** ***i :=*** <выражение> **;** <спис. операторов> **end**
⇒ **begin** ***i :=*** **read ;** <спис. операторов> **end**
⇒ **begin** ***i :=*** **read ;** <оператор>**;** <спис. операторов> **end**
⇒ **begin** ***i :=*** **read ;** ***i :=*** <выражение>**;** <спис. операторов> **end**
⇒ **begin** ***i :=*** **read ;** ***i :=*** **read ;** <спис.операторов> **end**
⇒ **begin** ***i :=*** **read ;** ***i :=*** **read ;** <оператор>**;** <спис.операторов> **end**
⇒ ... ...

Вывод и дерево вывода программы на языке Милан

```
begin
    m:=read;
    n:=read;
    while m≠n do
      if m>n then m:=m-n
      else n:=n-m;
    write(m)
end
```

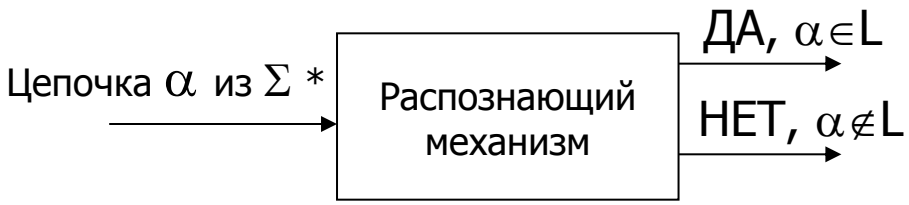


Вывод

<программа>
⇒ **begin** <спис. операторов> **end**
⇒ **begin** <оператор>**;** <спис. операторов> **end**
⇒ **begin i :=** <выражение> **;** <спис. операторов> **end**
⇒ **begin i := read ;** <спис. операторов> **end**
⇒ **begin i := read ;** <оператор>**;** <спис. операторов> **end**
⇒ **begin i := read ; i :=** <выражение>**;** <спис. операторов> **end** ⇒ … … …

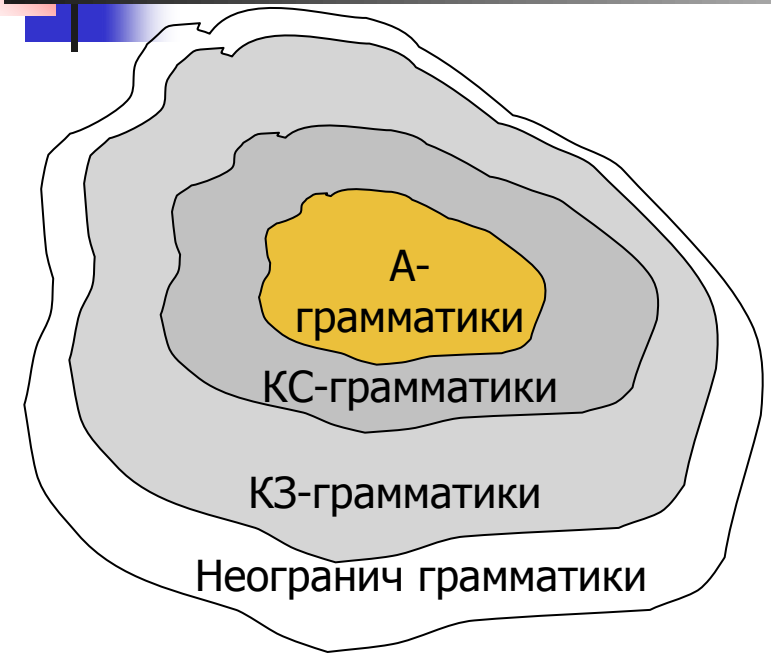
Дерево вывода

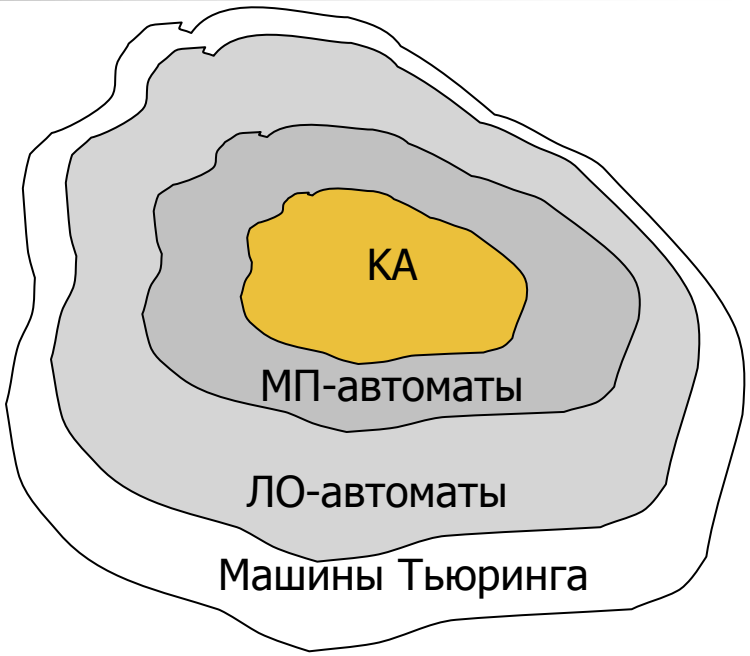
Проблема задания бесконечных множеств объектов



- Сложность описания языков состоит в том, что необходимо конечными средствами задать бесконечное множество объектов – цепочек языка
  - Существуют разные подходы к такому описанию. Один из таких подходов состоит в задании устройства для распознавания всех правильно построенных предложений
- Для строгой иерархии порождающих грамматик Хомского существует соответствующая строгая иерархия распознающих устройств
  - Особенностью этих устройств является то, что все они не в полной мере соответствуют понятию алгоритма: эти распознаватели недетерминированные. Иными словами, по распознавателю, соответствующему некоторой порождающей грамматике, в общем случае невозможно непосредственно построить алгоритм распознавания

Иерархия грамматик Хомского и распознающих автоматов





$\alpha \rightarrow \beta$	Неогранич грамм
$\alpha A \beta \rightarrow \alpha \gamma \beta$	КЗ - грамматики
$A \rightarrow \gamma$	КС - грамматики
$A \rightarrow aB$ $A \rightarrow a$ $A \rightarrow \varepsilon$	Автоматные грамматики (А- грамматики)

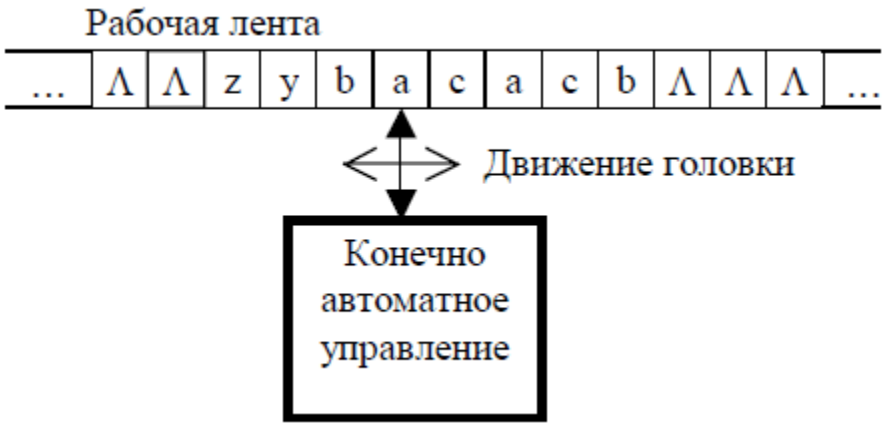
Существует иерархия распознающих автоматических устройств, полностью соответствующая иерархии порождающих грамматик Хомского

Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название грамматик	Распознающие абстрактные устройства
0	$\alpha \rightarrow \beta$	Неограниченные грамматики	Машины Тьюринга
1	$\gamma A \delta \rightarrow \gamma \beta \delta$	Контекстно-зависимые грамматики, Неукорачивающие грамматики, Грамматики непосредственных составляющих, НС-грамматики	Линейно-ограниченные автоматы
2	$A \rightarrow \beta$	Контекстно-свободные грамматики, КС-грамматики	Автоматы с магазинной памятью
3	$A \rightarrow aB$ $A \rightarrow a$ $A \rightarrow \varepsilon$	Автоматные грамматики А-грамматики Регулярные грамматики	Конечные автоматы

Машины Тьюринга

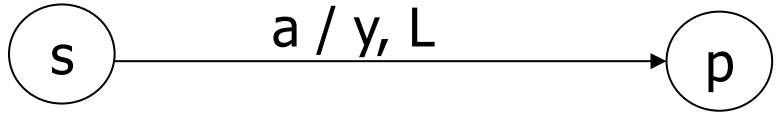
Алан Тьюринг в 1936г. предложил формальную модель вычислителя, которая является результатом простого добавления потенциально бесконечной памяти к системе управления с конечной памятью



Программа МТ – конечное множество пятерок

<s, a, p, y, D>, или

(s, a) ⟶ (p, y, D)



МТ = (S, Σ, s_0, δ, Г)

Г = {L, R, H};

δ: S × Σ ⟶ S × Σ × Г

Программа:

1. (s, a) ⟶ (p, y, L);
2. (q, b) ⟶ (s, z, R);
3. …

S – состояния

Σ - вход. алфавит

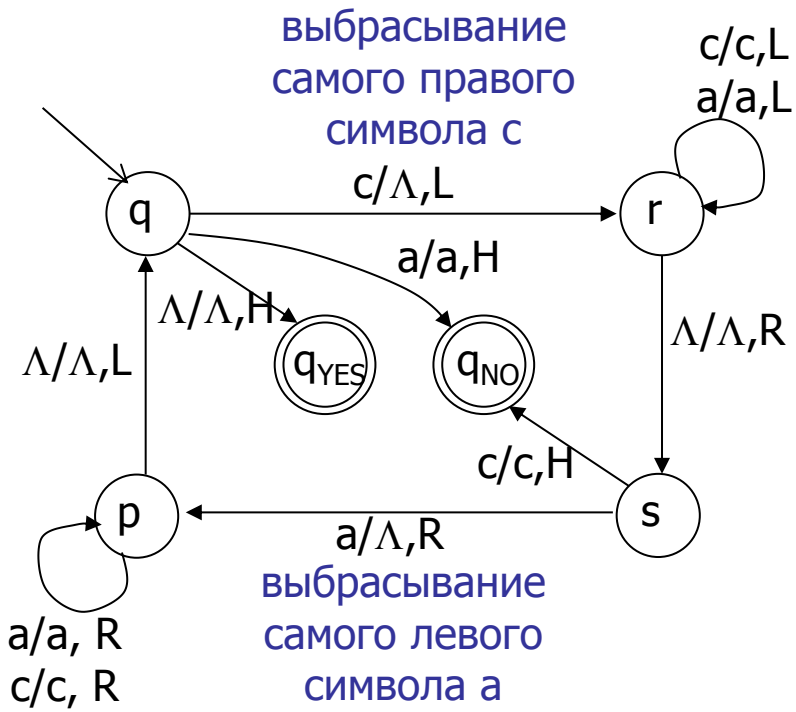
s_0 – начальное состояние

δ – функция переходов

Г – движение головки

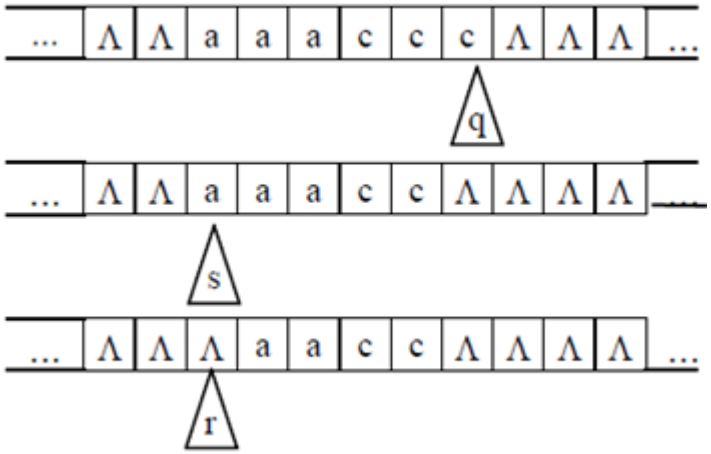
Машина Тьюринга, распознающая язык $a^n c^n$

Начальная конфигурация МТ: МТ находится в начальном состоянии q, а головка расположена против крайнего правого символа входной цепочки



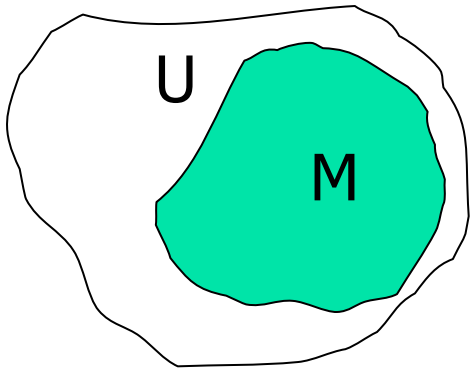
промежуточные конфигурации

начальная конфигурация:



Вопрос: какие ошибки во входной строке может обнаружить эта МТ?

Машина Тьюринга как распознаватель языка типа 0

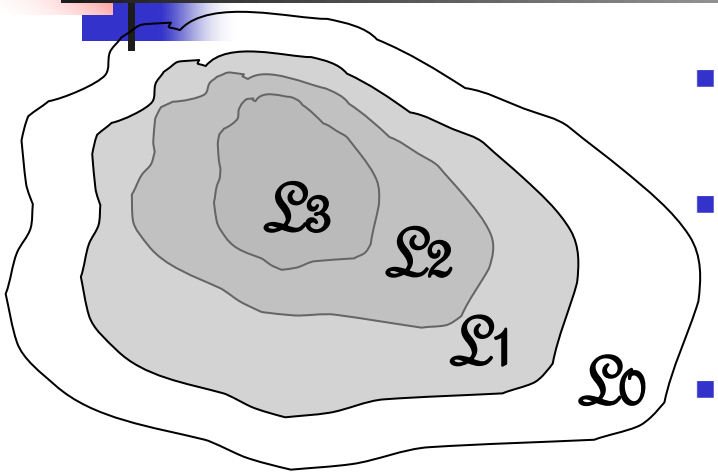


- Определение. Множество M называется рекурсивно-перечислимым, если существует машина Тьюринга, которая для любого входа x, если $x \in M$ перейдет в заключительное состояние q_{YES} и остановится , а если $x \notin M$, то либо перейдет в состояние q_{NO} и остановится, либо вообще не остановится

- Определение. Множество M называется рекурсивным (или разрешимым), если существует машина Тьюринга, которая для любого входа x, если $x \in M$ перейдет в заключительное состояние q_{YES} и остановится , а если $x \notin M$, то перейдет в состояние q_{NO} и остановится

- Для разрешимого множества существует алгоритм распознавания принадлежности его элементов. Перечислимость является более слабым вариантом эффективности, чем разрешимость. Если множество M не разрешимое, то алгоритма распознавания принадлежности ему элементов НЕ СУЩЕСТВУЕТ: для (рекурсивно-)перечислимого множества попытка выяснить, принадлежит ли ему некоторый элемент, может быть неудачной

Машина Тьюринга как распознаватель языка типа 0



- Язык – множество цепочек над конечным словарем Σ
- Распознавание языка L – это проверка того, является произвольная цепочка из множества Σ^* элементом множества L, либо нет.
- Распознаватель языка L – алгоритм определения принадлежности элемента множеству

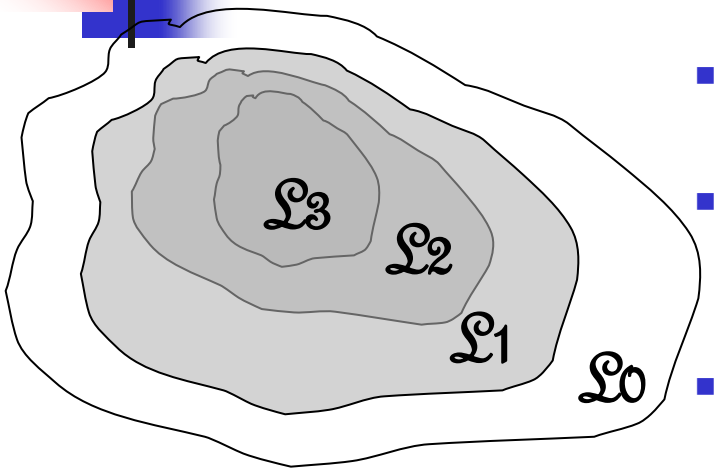
Теорема.

Любой язык, порождаемый грамматикой Хомского типа 0, является рекурсивно перечислимым . Это значит, что для любого формального языка существует машина Тьюринга, которая на цепочках, принадлежащих языку останавливается в состоянии q_{YES}, а на цепочках, НЕ принадлежащих языку – либо останавливается в состоянии q_{NO}, либо не останавливается

Следствие.

Проблема распознавания языков типа 0 алгоритмически неразрешима.
Не существует общего распознающего АЛГОРИТМА, который проверяет принадлежность произвольной цепочки языку, порождаемому произвольной грамматикой типа 0

Распознавание языков типа 1, 2, 3



- Язык – множество цепочек над конечным словарем Σ
- Распознавание языка L – это проверка того, является произвольная цепочка из множества Σ^* элементом множества L, либо нет.
- Распознаватель языка L – алгоритм определения принадлежности элемента множеству

Теорема.

Любой язык, порождаемый грамматикой Хомского типа 1 (и, следовательно, порождаемый грамматикой Хомского типа 2 и 3), является разрешимым . Это значит, что для любого формального языка типа 1, 2 и 3 существует машина Тьюринга (алгоритм), которая на цепочках, принадлежащих языку останавливается в состоянии q_{YES}, а на цепочках, НЕ принадлежащих языку – останавливается в состоянии q_{NO}

Следствие.

Проблема распознавания языков типов 1, 2 и 3 алгоритмически разрешима. Существуют общий распознающий алгоритм для языков типа 1 (КЗ-языков), и общий распознающий алгоритм для языков типа 2 (КС-языков), и общий распознающий алгоритм для языков типа 3 (А-языков).

Подклассы грамматик иерархии Хомского

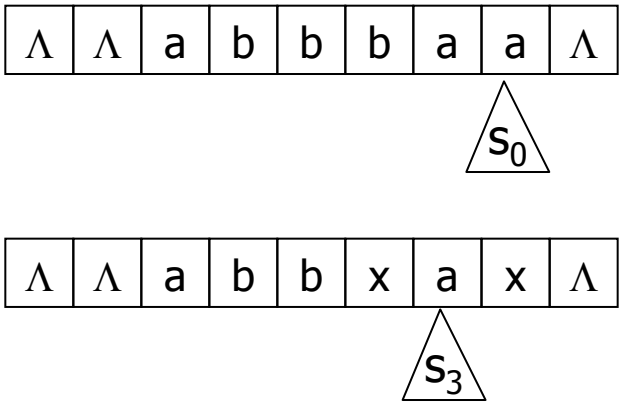
Тип	Вид правил	Название грамматик	Распознающие абстрактные устройства
0	$\alpha \to \beta$	Неограниченные грамматики	Машины Тьюринга
1	$\gamma A \delta \to \gamma \beta \delta$	Контекстно-зависимые грамматики, Неукорачивающие грамматики, Грамматики непосредственных составляющих, НС-грамматики	Линейно-ограниченные автоматы
2	$A \to \beta$	Контекстно-свободные грамматики, КС-грамматики	Автоматы с магазинной памятью
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики А-грамматики Регулярные грамматики	Конечные автоматы

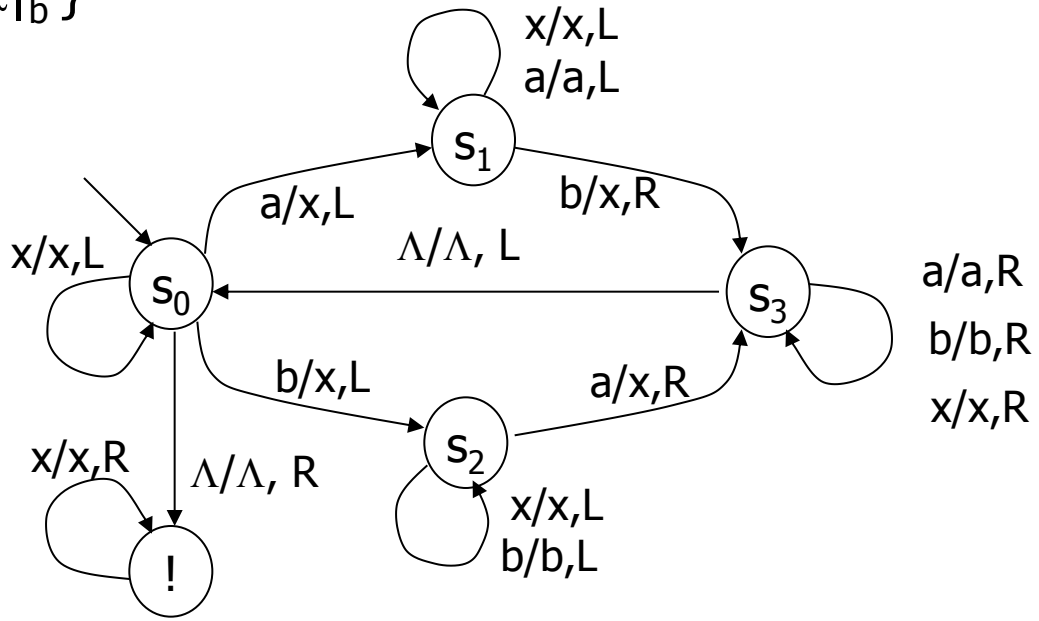
Линейно-ограниченные автоматы

- Машина Тьюринга называется линейно-ограниченным автоматом, если существует такое число C, что в процессе обработки входной ленты, на которой в начальной конфигурации находится цепочка ω, машина не может использовать более, чем C| ω | ячеек ленты. Таким образом, объем памяти такого автомата является ограниченным и определяется длиной входной цепочки, в отличие от потенциально неограниченной памяти у машины Тьюринга
- **Теорема.**
  Множество цепочек над конечным словарем является языком типа 1 (контекстно-зависимым) тогда и только тогда, когда оно является множеством цепочек, распознаваемых некоторым недетерминированным линейно-ограниченным автоматом.

Пример: линейно-ограниченный автомат

- Построим линейно-ограниченный автомат, распознающий язык, в цепочках которого одинаковое число вхождений символов a и b: $L=\{\alpha \in \{a, b\}^* \mid |\alpha|_a = |\alpha|_b\}$





Число ячеек на ленте при распознавании цепочки языка не возрастает.

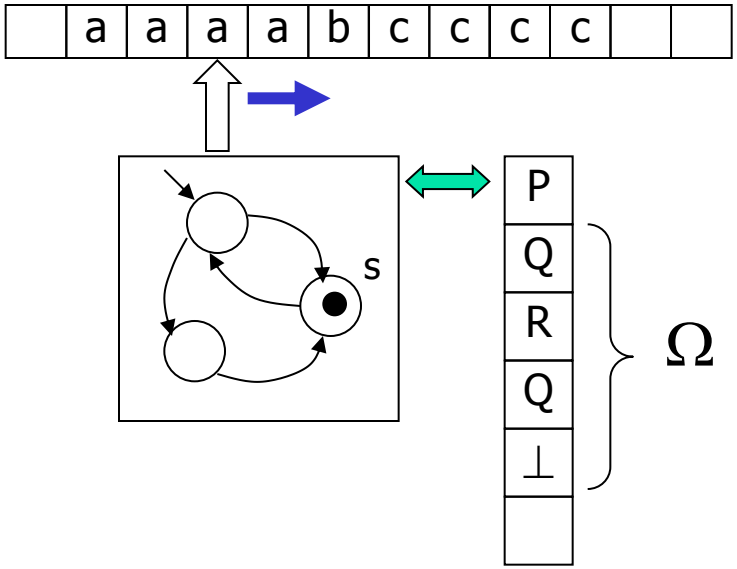
Следствие: Эта МТ – линейно ограниченный автомат, и этот язык может быть порожден контекстно-зависимой грамматикой Хомского

Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название грамматик	Распознающие абстрактные устройства
0	$\alpha \to \beta$	Неограниченные грамматики	Машины Тьюринга
1	$\gamma A \delta \to \gamma \beta \delta$	Контекстно-зависимые грамматики, Неукорачивающие грамматики, Грамматики непосредственных составляющих, НС-грамматики	Линейно-ограниченные автоматы
2	$A \to \beta$	Контекстно-свободные грамматики, КС-грамматики	Автоматы с магазинной памятью
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики А-грамматики Регулярные грамматики	Конечные автоматы

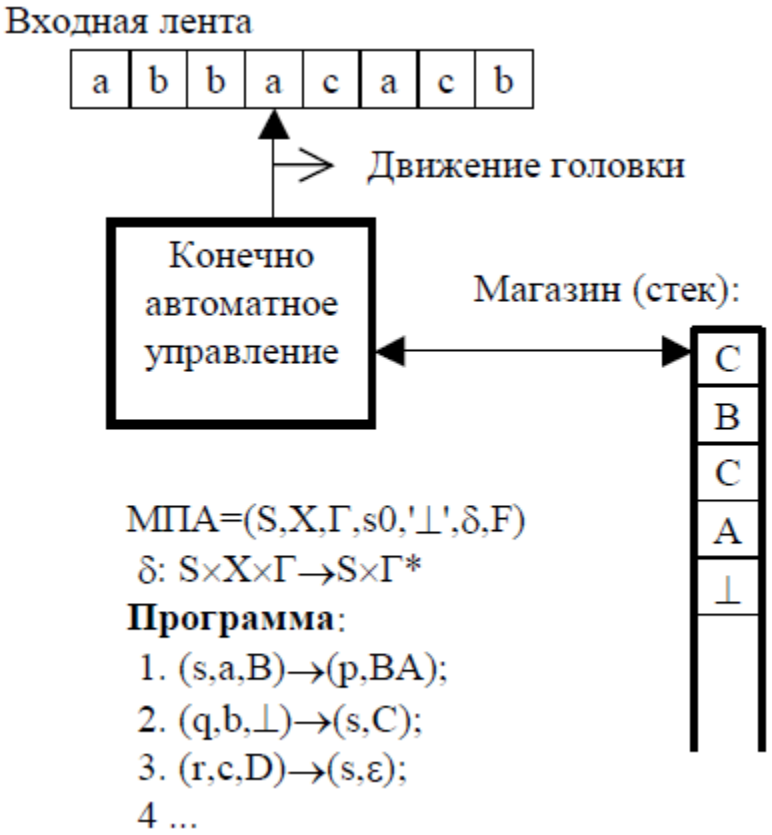
Автоматы с магазинной памятью и контекстно-свободные грамматики

- Распознавателями КС-языков является класс автоматов с магазинной памятью (МП - автоматами)
- МП-автомат – конечный автомат, снабженный дополнительным стеком (магазином) для хранения промежуточной информации потенциально бесконечного объема. У МП-автомата конечное число состояний S с начальным состоянием q_0 и подмножеством F финальных (допускающих) состояний, конечный алфавит входных сигналов Σ и конечный алфавит магазина с маркером 'дно стека' $\perp$.



- МП-автомат $M = (S, \Sigma, \Gamma, \delta, q_0, \perp, F)$
  S – множество состояний,
  Σ – входной алфавит,
  Γ – алфавит магазина,
  $\delta: S\times(\Sigma\cup\{\varepsilon\})\times\Gamma \to 2^S\times\Gamma^*$,
  $q_0 \in S$ – начальное состояние,
  $F \subseteq S$ – допускающие состояния,
  $\perp \in \Gamma$ - дно магазина

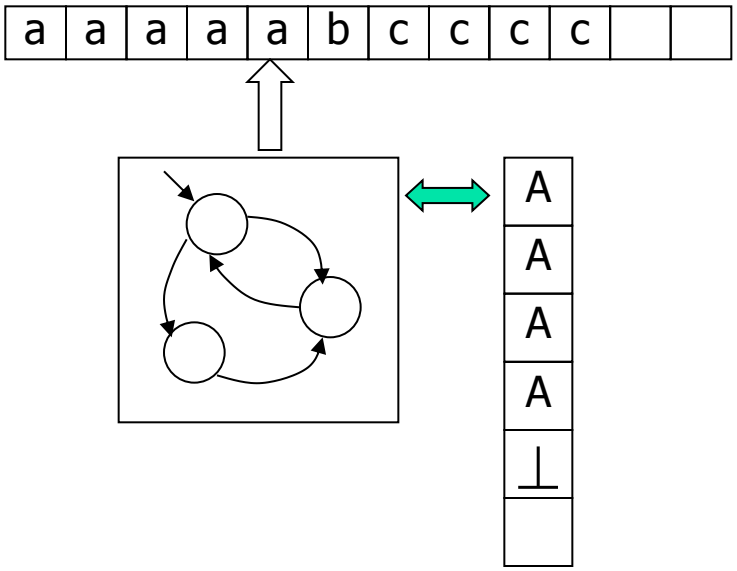
Структура МП-автомата



Функция переходов δ по каждой тройке *<текущее состояние, очередной входной сигнал, верхний символ магазина>* определяет на очередном шаге функционирования следующее состояние и цепочку символов магазинной памяти, записываемых в магазин вместо прочитанного верхнего символа (цепочка записывается в магазин "хвостом вперед", т.е.сначала записывается последний символ цепочки, затем предпоследний, и т.д.).

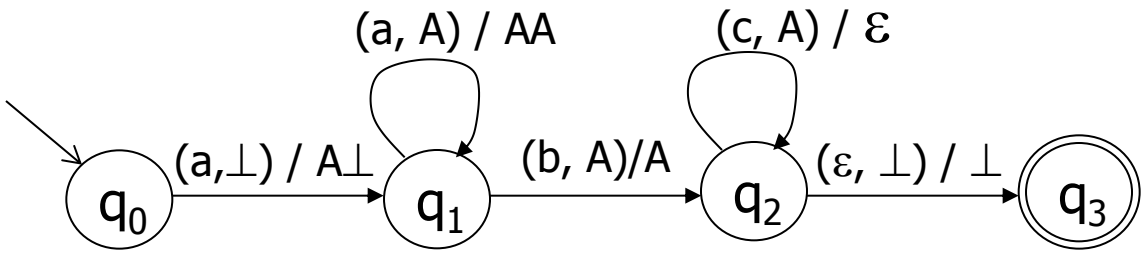
МП-автомат распознает входную цепочку, если по ее завершении автомат перейдет в одно из заключительных состояний и его магазин будет пустым

Пример: МП-автомат, распознающий язык $\{a^n b c^n | n>0\}$

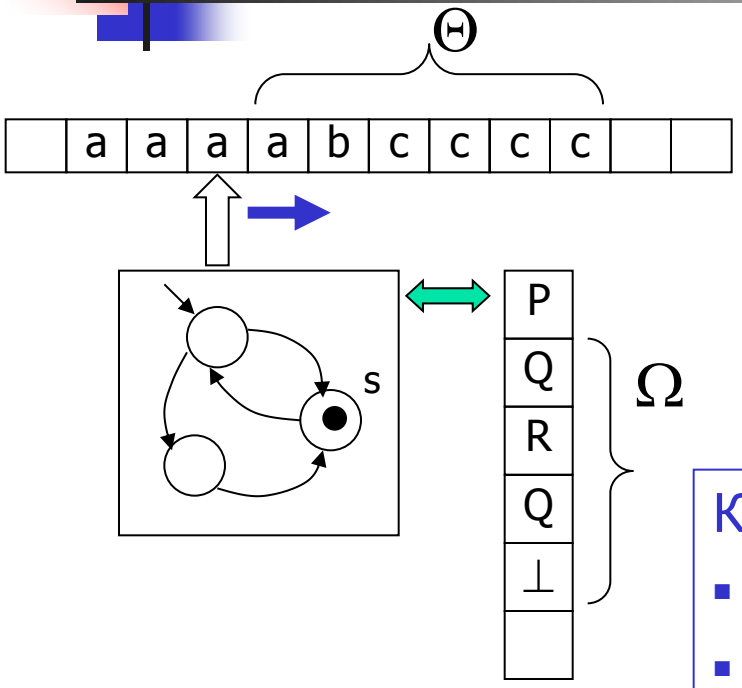


$S = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{ a, b, c\}$
$\Gamma = \{A, \perp\}$
$\delta: S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^S \times \Gamma^*$
$q_0 \in S$
$F = \{q_3\}$
$\perp \in \Gamma$

Функция переходов δ :



Такт работы МП-автомата



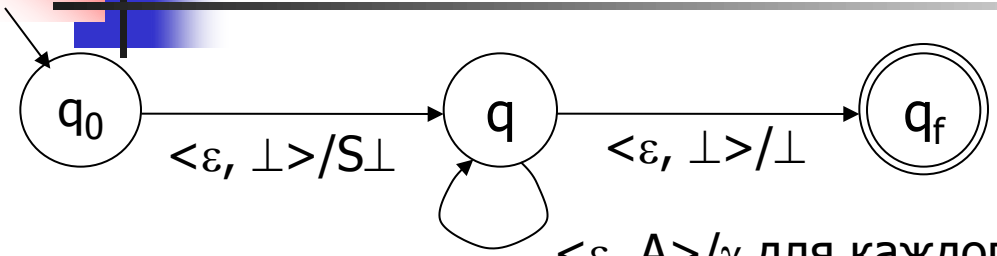
- МП-автомат M = (S, Σ, Γ, δ, q_0, ⊥, F)
  - S – множество состояний,
  - Σ – входной алфавит,
  - Γ – алфавит магазина,
  - $\delta: S\times(\Sigma\cup\{\varepsilon\})\times\Gamma \to 2^S\times\Gamma^*$,
  - $q_0 \in S$ – начальное состояние,
  - $F \subseteq S$ – допускающие состояния,
  - $\perp \in \Gamma$ - дно магазина

Конфигурация: <s, aΘ, PΩ>:

- s – текущее состояние
- Θ – непрочитанная часть цепочки на входной ленте
- $P \in \Gamma$ – верхний символ магазина
- $\Omega \in \Gamma^*$ - остаток цепочки, находящейся в магазине

- Такт (шаг) работы: <s, aΘ, PΩ > |- < s′, Θ, γΩ >, если $(s', \gamma) \in \delta(s, a, P)$
- Пустой такт <s, aΘ, PΩ> |- <s′, aΘ, γΩ>, если $(s', \gamma) \in \delta(s, \varepsilon, P)$
- Цепочка Φ допускается, если $\langle s_0, \Phi, \perp\rangle \vdash^* \langle q_{YES}, \varepsilon, \perp\rangle$, где $q_{YES} \in F$

МП автомат для распознавания КС-языка



<ε, A>/γ для каждого правила грамматики $A \to \gamma$

<a, a>/ε для каждого терминала $a \in \Sigma$

Функция переходов δ искомого МП-автомата строится так:

- $\delta(q_0, \varepsilon, \perp)=(q, S \perp)$ // в верхушку магазина помещаем начальный нетерминал грамматики G
- $(\forall a \in T)$: $\delta(q, a, a)=(q,\varepsilon)$ // если в верхушке магазина – терминальный символ a, и очередной входной символ цепочки a, то автомат переходит к следующему входному символу, выбрасывая из магазина верхний символ
- $(\forall a \in T)\ (\forall A \in N)(\forall A \to \gamma \in R)$: $\delta(q, \varepsilon, A)=(q, \gamma)$ // если в верхушке магазина стоит нетерминал, то МП-автомат, не читая очередного входного символа, недетерминированно заменяет его в магазине одной из альтернатив этого нетерминала из множества R продукций грамматики G
- $\delta(q, \varepsilon, \perp)=(q_f, \perp)$ // если в магазине больше нет символов, то переходим в заключительное состояние, не меняя магазин

Построенный так МП-автомат (недетерминированно) моделирует процесс левостороннего порождения цепочки языка в грамматике G при удачном выборе последовательности выполнения команд

Свойства МП автоматов

- Теорема. По каждой КС-грамматике G=(N,T,S,R) можно построить (недетерминированный) МП - автомат M_G, распознающий язык, порождаемый G.
- Множество состояний M_G составляют три состояния: начальное состояние q_0, рабочее состояние q и заключительное состояние q_f. Множество входных сигналов – это множество терминальных символов грамматики G.
- Для простоты будем считать, что алфавит магазина совпадает с объединенным алфавитом терминальных и нетерминальных символов
- **Теорема.**
  Язык палиндромов $\{ww^R|w\in\{0,1\}\}$ не может быть задан никаким детерминированным МП-автоматом, но может быть задан недетерминированным МП-автоматом (как язык типа 2)
- Иными словами, недетерминированные МП-автоматы могут распознавать более широкий класс языков, чем детерминированные МП-автоматы. В этом их существенное отличие от КА

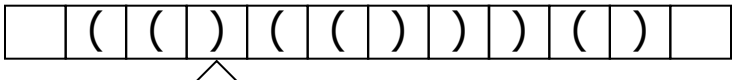
МП-автомат для распознавания правильных скобочных выражений по грамматике: S→ SS | (S) | ε

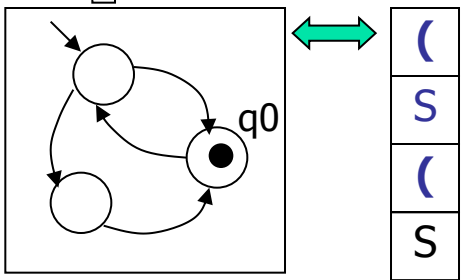
- Множество состояний – начальное состояние q_0, рабочее состояние q и заключительное состояние q_f.
- Множество входных сигналов – множество терминалов грамматики G
- Магазинный алфавит включает единственный нетерминал грамматики S и пару символов, ‘(‘ и ‘)’ - терминалы грамматики
- M =({ q_0, q, q_f}, {(,)}, {S, (,)}, q_0, ⊥, δ, {q_f})
- Функция переходов δ этого МП-автомата определится так:
  δ(q_0, ε, ⊥)=(q, S⊥); // в магазин -> начальный нетерминал грамматики;
  δ(q, (, ( )=(q, ε); // если терминал входной цепочки и верхушка
  // магазина совпадают, выбрасываем символ из магазина;
  δ(q,),))=(q, ε); // ---------- " ---------.
  δ(q, ε, S)=(s, SS); // в соответствии с продукцией грамматики;
  δ(q, ε, S)=(s, (S)); // ---------- " ---------;
  δ(q, ε, S)=(s, ε); // ---------- " ---------;
  δ(q, ε, ⊥)=(q_f, ⊥); // при пустом магазине входная цепочка распознана, если
  // она кончилась

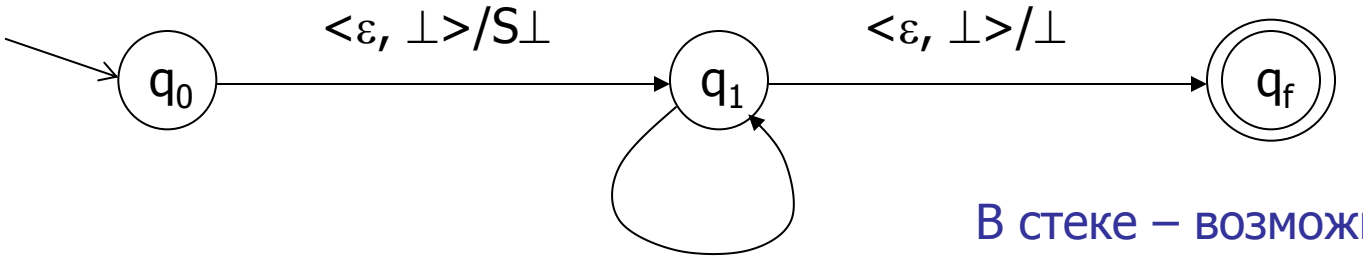
Алгоритм распознавания языка очень непросто построить по такому недетерминированному автомату.

Недетерминированный МП-автомат для распознавания скобочных выражений по грамматике: S→ SS | (S) | ε

1. S→ SS
2. S → (S)
3. S → ε







Если в верхушке стека S, то в стек вместо S помещаем возможные правые части правил с S в левой части

<(, (>/ ε
<),)>/ ε
<ε, S>/ S S
<ε, S>/(S)
<ε, S>/ ε

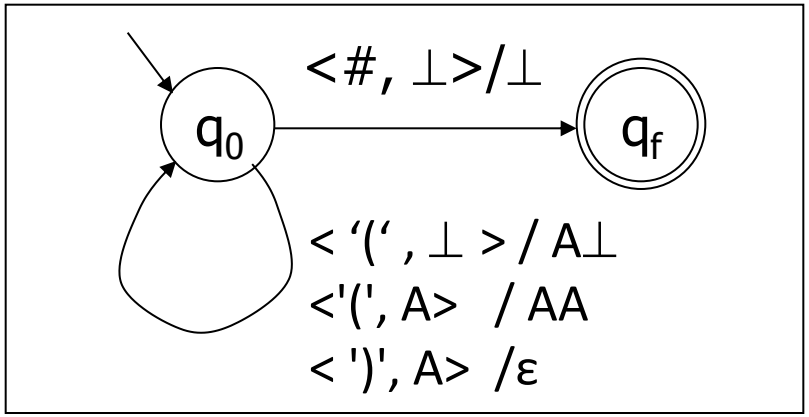
В стеке – возможные промежуточные цепочки, выводимые из начального символа S

Пример: детерминированный МП-автомат для языка правильных скобочных выражений

Можно построить распознающий МП-автомат не по грамматике

Детерминированный МП-автомат для распознавания этого языка
Вначале стек пуст – в нем находится заключительный маркер '⊥`. По каждой встретившейся на входе открывающей скобке '(' автомат добавляет в магазин символ A (который отмечает, что соответствующая закрывающая скобка должна впоследствии встретиться во входной цепочке). Каждая встретившаяся на входе закрывающая скобка приводит к выбрасыванию символа A из магазина. Когда все символы и во входной строке, и в стеке исчерпываются, автомат переходит в заключительное состояние

строка:
(( ) ( ) (( )))#

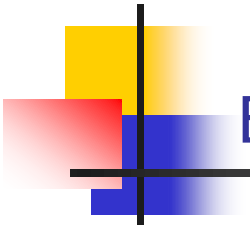


Любая несбалансированность скобок не может привести к ситуации, когда при закончившейся входной строке стек пуст

Дополнительные требования к языку и КС-грамматики

- Многие тонкие свойства языка не выразимы посредством КС-грамматики
- Чтобы выразить в грамматике дополнительные требования:
  - обязательное объявление имени до его использования в программе
  - требование соответствия типов левой и правой части оператора присваивания

  необходимо наложить дополнительные ограничения на вид правил
- КС-грамматикой этого сделать невозможно
- Для любой КС-грамматики существует МП-автомат, генерирующий или распознающий тот же язык, что и данная КС-грамматика. Для проверки того, объявлено или нет данное имя в программе, необходимо просмотреть по программе множество объявлений, т.е. вернуться назад по тексту программы. Магазин для этого использовать не получится.
- В случае грамматик типа 1 мы имеем линейно-ограниченный автомат, Машину Тьюринга, в которой символы записываются на ленте, по которой, в случае необходимости, мы можем двигаться и в обратном направлении, не выталкивая символы, как это приходится делать в магазине. В рамках этой техники мы сможем проверить объявление имени



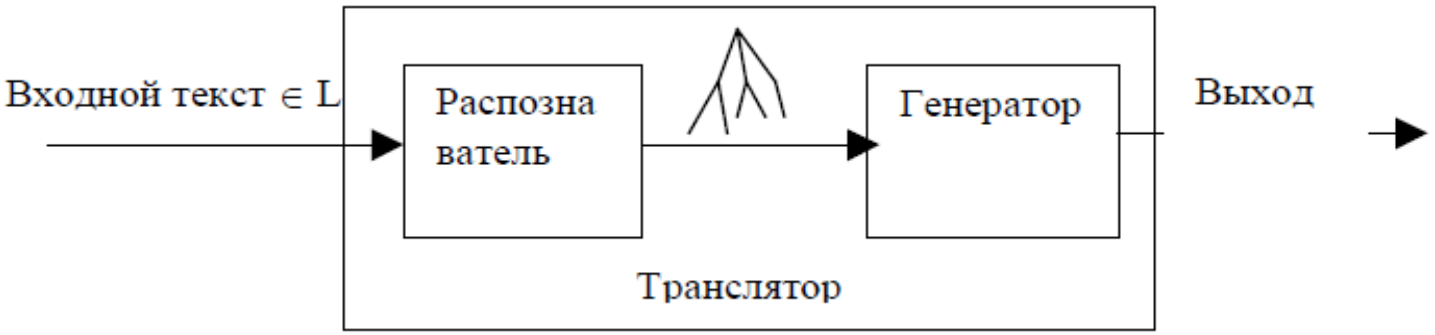
Выводы

- Для любой КС-грамматики G = (Σ, N, S, R) можно построить МП-автомат, множество допускаемых цепочек которого будет совпадать с языком, порождаемым грамматикой
- Недетерминированный МП-автомат распознает входную цепочку, если существует последовательность шагов его работы, приводящая его из начального в одно из заключительных состояний после прочтения всей входной цепочки и пустом магазине
- Класс языков, распознаваемых недетерминированными МП-автоматами совпадает с классом языков, порождаемых грамматиками типа 2 (КС-грамматиками). Поэтому МП-автомат может быть использован для представления любого языка типа 2.
- Очевидно, что по любому МП-автомату может быть построена машина Тьюринга, моделирующая его функционирование
- По недетерминированному МП-автомату, распознающему язык L, всегда можно построить АЛГОРИТМ, распознающий тот же язык L

Подклассы грамматик иерархии Хомского

Тип	Вид правил	Название грамматик	Распознающие абстрактные устройства
0	$\alpha \to \beta$	Неограниченные грамматики	Машины Тьюринга
1	$\gamma A \delta \to \gamma \beta \delta$	Контекстно-зависимые грамматики, Неукорачивающие грамматики, Грамматики непосредственных составляющих, НС-грамматики	Линейно-ограниченные автоматы
2	$A \to \beta$	Контекстно-свободные грамматики, КС-грамматики	Автоматы с магазинной памятью
3	$A \to aB$ $A \to a$ $A \to \varepsilon$	Автоматные грамматики А-грамматики Регулярные грамматики	Конечные автоматы

Распознавание автоматных языков



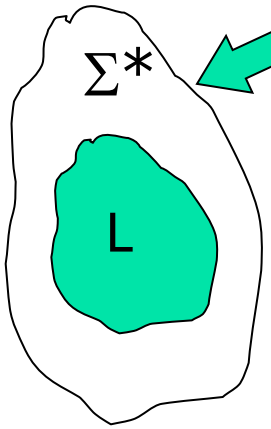
- Если язык порождается автоматной грамматикой Хомского, то для распознавания по грамматике нужно построить соответствующий конечный автомат
- Если этот конечный автомат недетерминированный, то нужно построить эквивалентный детерминированный конечный автомат
- Детерминированный конечный автомат-распознаватель и будет представлять алгоритм распознавания, т.е. восстановления вывода любой цепочки
- Этот алгоритм очень эффективный

Автоматные языки, автоматные грамматики, регулярные выражения, ...

Регулярное выражение – описание формулой

Недетерминированный КА описание графом переходов

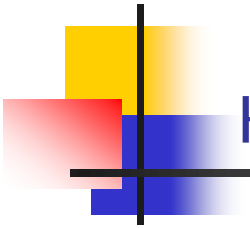
Автоматная порождающая грамматика



L -автоматный язык (регулярное множество бесконечного числа цепочек символов)

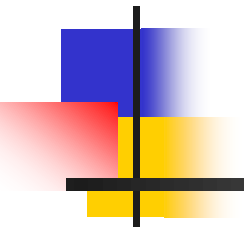
Детерминированный КА

Распознающий алгоритм



Конечные автоматы и грамматики типа 3

- Между автоматными порождающими грамматиками Хомского (грамматиками типа 3) и конечными автоматами существует тесная связь, а именно, для любого конечного автомата, допускающего язык L, существует автоматная грамматика, порождающая L, и наоборот
- Из-за такой связи языки типа 3 Хомский назвал языками «с конечным числом состояний».



Другие формализмы для задания языков

Нотация Бэкуса-Наура

- Для формального задания порождающих грамматик формальных языков также необходим язык. Этот язык иногда называется *метаязыком*
- Исторически первым метаязыком является нотация Бэкуса-Наура, или БНФ - нотация, которая названа так в честь Дж.Бэкуса (который был главным разработчиком первого компилятора для языка Фортран) и Питера Наура (который играл основную роль в разработке одного из первых алгоритмических языков – Алгола-60)
- БНФ-нотация использовалась впервые для строгого определения Алгола-60 еще до изобретения порождающих КС-грамматик Хомского
  - Нетерминалы помещаются в угловые скобки. Они называются “металингвистическими переменными”
  - Символ '::=' используется вместо стрелки "→"
  - Альтернативы одного и того же нетерминала записываются в правой части одного правила через знак '|', имеющий смысл "либо”
- Пример:

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

БНФ нотация

- В БНФ часто используются:
  - [ ] необязательные элементы: повторение 0 или 1 раз;
  - { } повторяющиеся элементы: 0 или произвольное число раз;
  - [ ]* повторяющиеся элементы: 0 или произвольное число раз
- Пример:

  <целое число> ::= [+ | –] <цифра> { <цифра> }

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

  <программа> ::= <заголовок>[<объявления >] begin {<инструкции>} end

  <заголовок> ::= program <идентификатор> **;**
- Терминальные символы – это символы не в угловых скобках

- Пример грамматики в БНФ-нотации:

  <Sentence> ::= <Name> [{, <Name> } <End>]

  <End> ::= and <Name>

  <Name> ::= tom | dick | barry

Примеры выводимых цепочек

tom;

barry and tom;

dick, barry, tom, and barry

dick, dick and dick

Примеры грамматик в нотации БНФ

В некоторых вариантах нотации БНФ нетерминалы записываются одним словом без скобок, а терминалы записываются в кавычках: ‘a’

Грамматика, порождающая операции преобразования множеств, например
A := {b, c} ; B := A ∩ {c, d} ∪ {a,b}

```
Program    ::= Statement { ';' Statement } '.'
Statement  ::= Name ':=' Expression
Expression ::= Expr  Op  Expr | Name | '{' Elements '}' | '(' Expr ')'
Op         ::= '∩' | '∪'
Name       ::= 'A' | 'B' | …| 'X' | 'Y' | 'Z'
Elements   ::= Element { ',' Element }
Element    ::= 'a' | 'b' | … | 'x' | 'y' | 'z'
```

Грамматика, порождающая группы печати страниц

```
Прог    ::= Группа { ',' Группа } 'eot'
Группа  ::= Целое [ '-' Целое ]
Целое   ::= Цифра { Цифра }
Цифра   ::= '0'|'1'| '2' | '3' | '4' | '5' | '6'| '7' | '8' | '9'
```

Pages: 23, 13-54, 128-97

Задание языков: БНФ и КС-грамматики Хомского

- Любую КС-грамматику Хомского можно задать с помощью БНФ . Пример:
  - <программа> ::= begin <последовательность операторов> end
  - <послед операторов> ::= <оператор>
  - <послед операторов> ::= <оператор> ; <послед операторов>
- Обратно: любую грамматику в форме БНФ можно задать КС-грамматикой :
  - Продукция БНФ A ::= α{β}γ эквивалентна трем продукциям КС-грамматики Хомского:
    $A \to \alpha B \gamma$, $B \to B \beta$, $B \to \varepsilon$
  - продукция БНФ A::=α[β]γ
    эквивалентна двум продукциям КС-грамматики Хомского
    $A \to \alpha \beta \gamma$ и $A \to \alpha \gamma$
- Расширения упрощают запись:

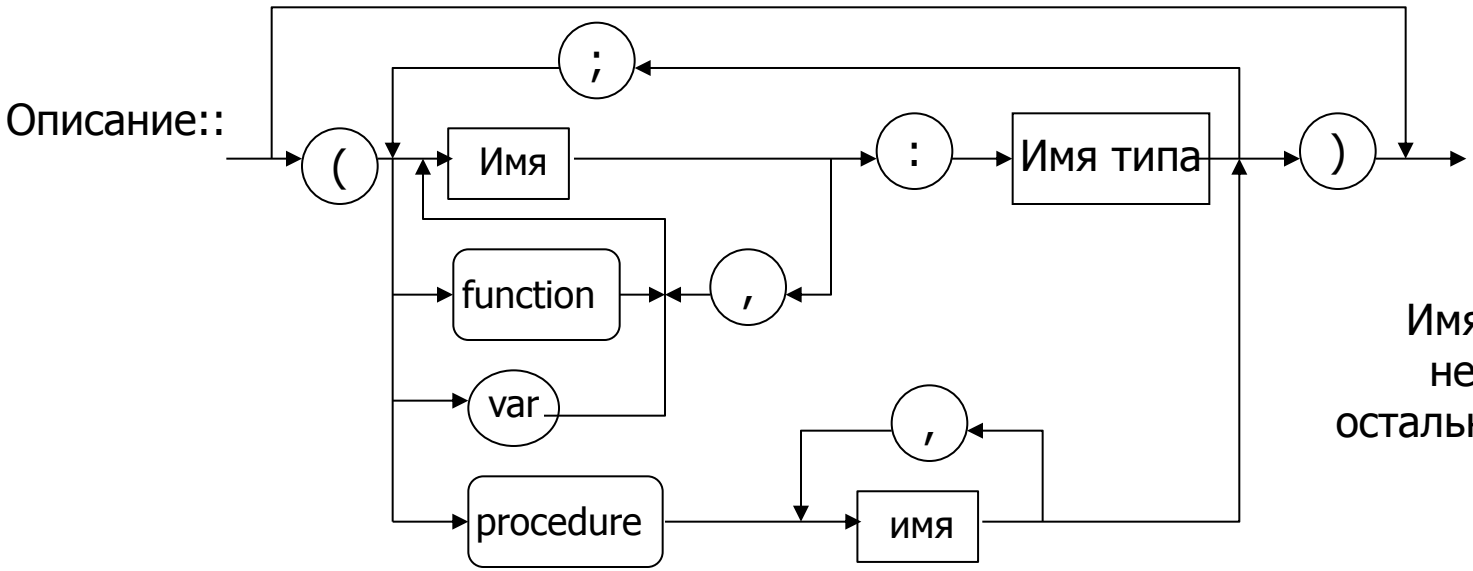
<variable-declaration-part> ::= var <var-declaration> {; <var-declaration>}

<integer-constant> ::= [+ | -] <digit> {<digit>}

- Итак, БНФ-нотация – это просто удобная форма записи правил КС-грамматик Хомского как регулярных выражений (а не цепочек символов)
- Существуют некоторые различия в разных форматах записи БНФ

Язык синтаксических диаграмм

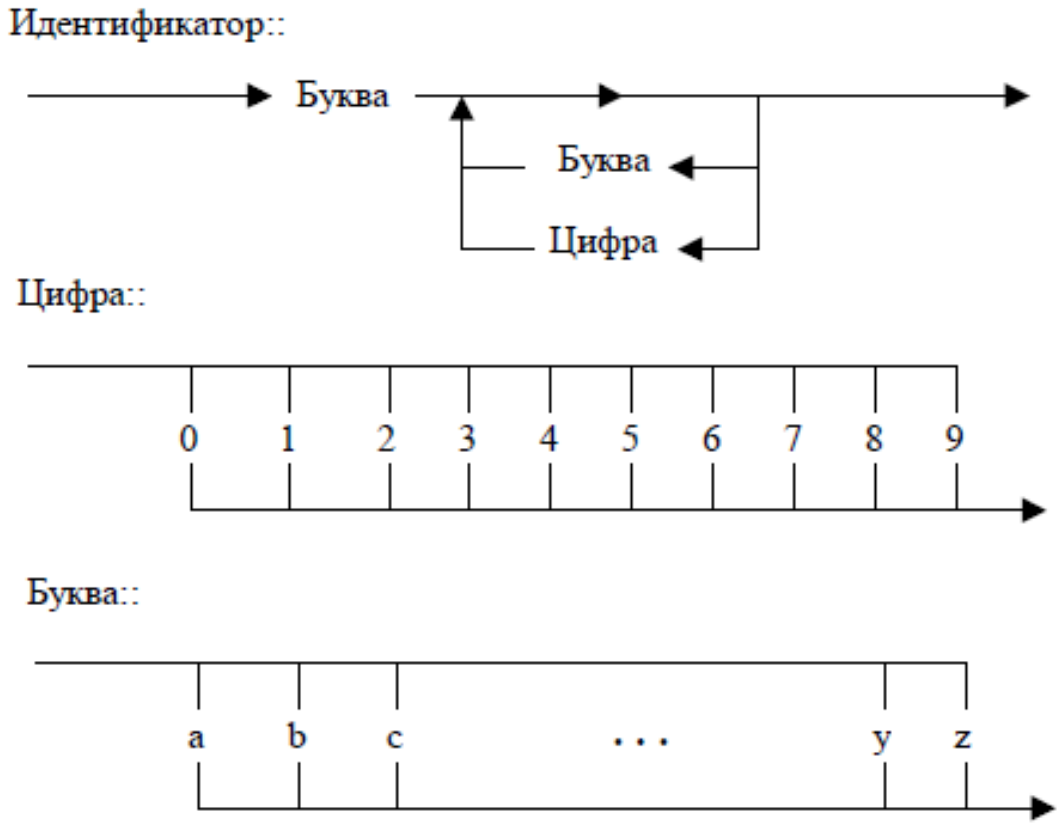
- Синтаксические диаграммы служат для определения языков с помощью графического представления. Были введены Виртом для задания языка Паскаль, использовались для задания языков Модула-2 и Фортран 77 и др.
- Синтаксические диаграммы являются порождающим механизмом. Правила порождения цепочек языка часто представлены в виде нескольких синтаксических диаграмм
- Любой путь в диаграмме от входа к выходу задает возможную последовательность символов в порождаемой цепочке языка



Имя, Имя типа – нетерминалы; остальные - терминалы

Определение идентификатора синтаксическими диаграммами

Каждому нетерминалу в задании грамматики соответствует своя диаграмма.
Встречающиеся нетерминалы должны быть заменены произвольной цепочкой, определяемой диаграммой, помеченной этим нетерминалом.
Например, идентификатор в этой нотации может определяться так:



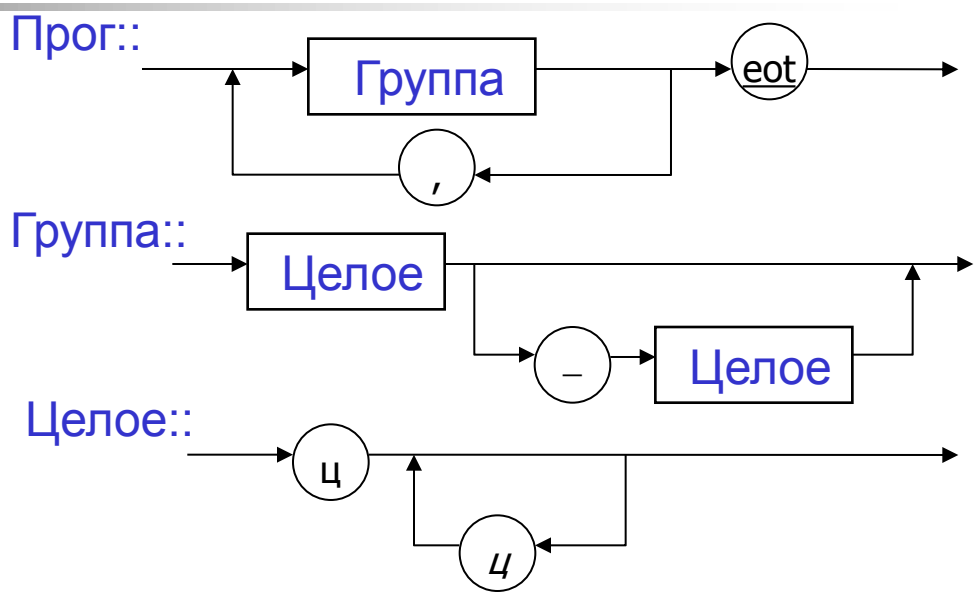
Синтаксические диаграммы языка перечисления номеров страниц

Pages: 23, 13-54, 128-97

Цепочка - это:
Группы, разделенные запятыми

Группа - это:
Целое, либо Целое тире Целое

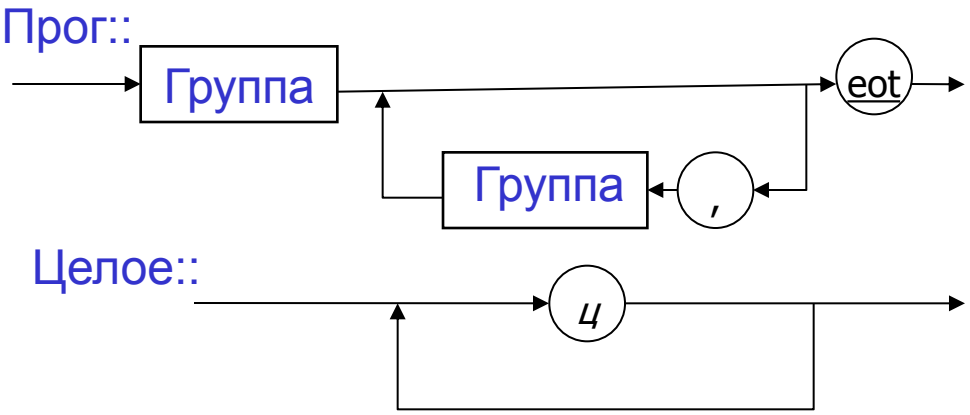
Целое – это:
непустая последовательность цифр



Грамматика:

Прог → Группа { , Группа } eot
Группа → Целое [- Целое]
Целое → ц { ц }

Можно и по-другому:

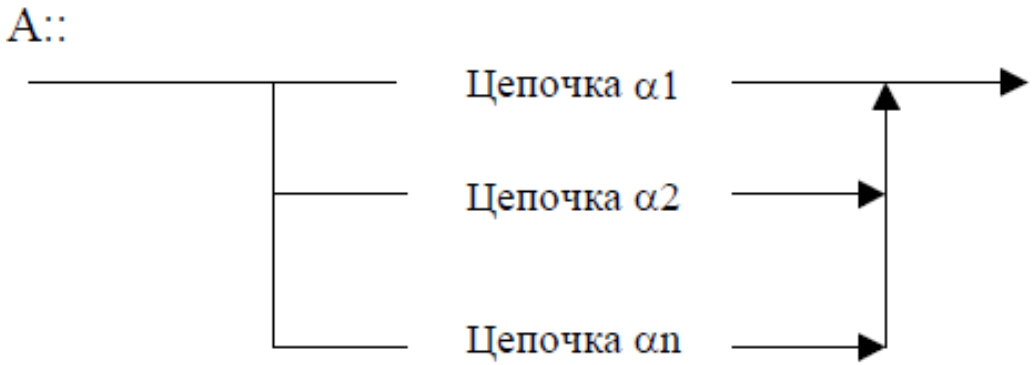


Равномощность синтаксических диаграмм классу КС грамматик Хомского

- Синтаксические диаграммы равномощны классу КС-грамматик Хомского. Иными словами, класс языков, порождаемых синтаксическими диаграммами, совпадает с классом КС-языков.
- Любая КС-грамматика может быть представлена эквивалентным набором синтаксических диаграмм. Действительно, сопоставим каждому множеству правил КС-грамматики G с одним и тем же нетерминалом в левой части вида

$$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$$

одну синтаксическую диаграмму с n разветвлениями, по одному для каждой альтернативы нетерминала A:



Очевидно, что так определенное множество синтаксических диаграмм порождает тот же язык, что и грамматика G

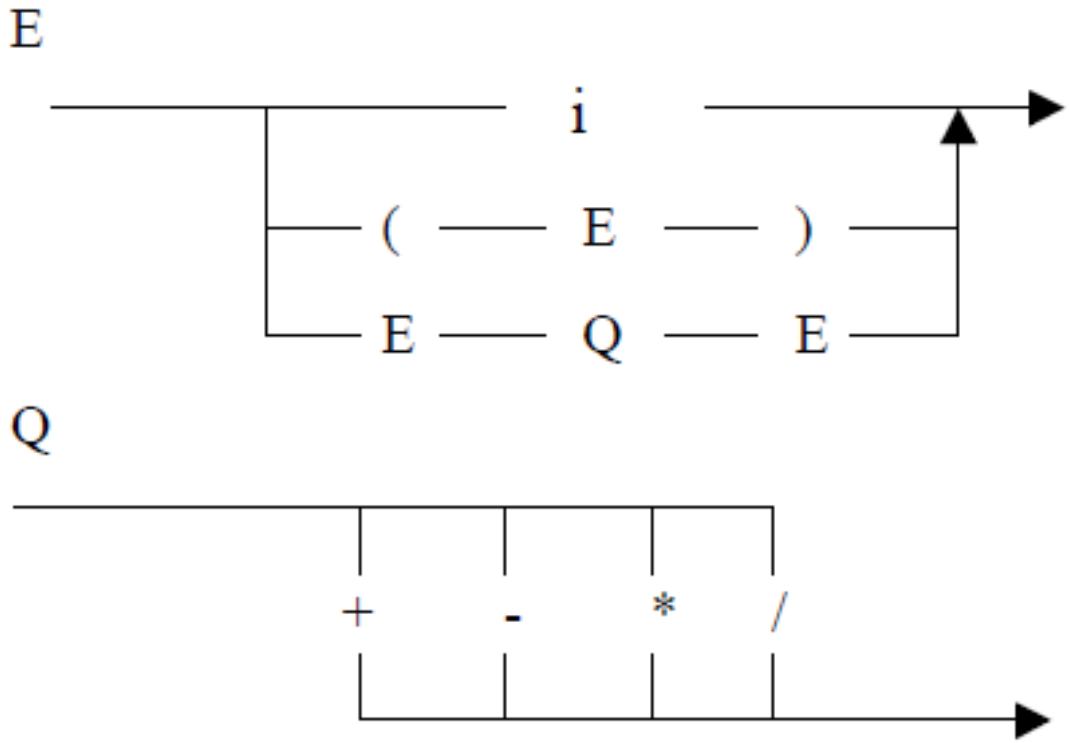
Пример

Грамматике

G: $E \rightarrow i \mid (E) \mid EQE$

$Q \rightarrow + \mid - \mid * \mid /$

соответствует следующее эквивалентное множество синтаксических диаграмм:



Построение эквивалентной КС – грамматики по синтаксическим диаграммам

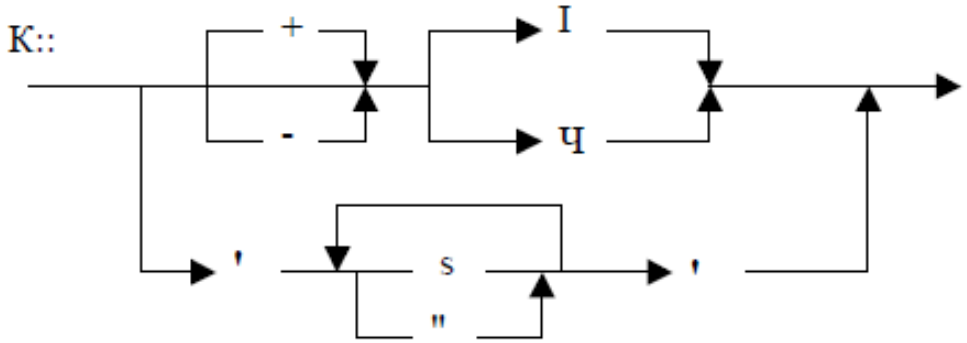
- Докажем обратное: по любому набору синтаксических диаграмм можно построить эквивалентную КС-грамматику
  1. Синтаксическую диаграмму можно считать конечным автоматом, состояния которого соответствуют ребрам синтаксической диаграммы, а переходы – соответствующим символам
  2. Поскольку любой конечный автомат допускает регулярный язык, то и синтаксическую диаграмму можно представить регулярным выражением над словарем, включающим символы – терминальные и нетерминальные, которые помечают ребра диаграммы
  3. Это означает, что каждая синтаксическая диаграмма может быть записана в БНФ нотации, и, следовательно, в эквивалентной форме КС-грамматики

Пример

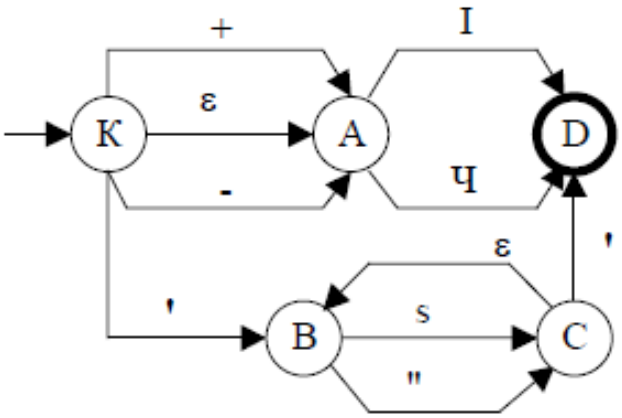
Пусть синтаксическая диаграмма задает константу в Паскале. Построим по ней конечный автомат, в котором кроме терминалов, могут встречаться и нетерминалы

Здесь обозначены

К - константа,
I - идентификатор константы,
Ч – число без знака,
s - символ.

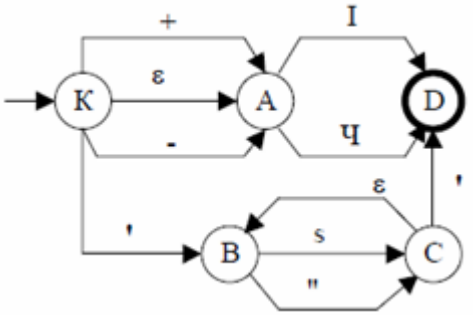


Синтаксическая диаграмма константы в Паскале



Соответствующий конечный автомат

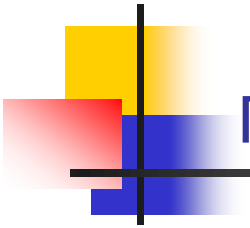
Пример (продолжение)



- БНФ-нотация для константы легко записывается прямо по диаграмме или автомату:
  K::=[+ | −] (I | Ч) | '(s | ") {s | "}`
- По расширенному конечному автомату может быть легко построена и эквивалентная КС-грамматика. Пометим все состояния автомата нетерминалами, причем начальное состояние помечается нетерминалом, соответствующим имени синтаксической диаграммы. D-заключительное состояние.
- Искомая КС-грамматика :

  G: K→ +A | -A | A | ' B
  A→ ID | ЧD
  B → sC | "C
  C → B | ' D
  D → ε

Синтаксические диаграммы – это графическая форма спецификации, фактически, аналогичная обычным порождающим контекстно-свободным грамматикам Хомского

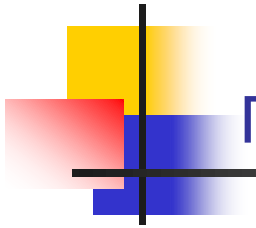


Грамматики с рассеянным контекстом

- Грамматики с рассеянным контекстом являются обобщением грамматик Хомского, цель их введения– упростить описание контекстных свойств естественных и искусственных языков (в частности, языков программирования)
- КС-грамматики Хомского слишком слабы, чтобы описать такие свойства языков программирования, как запрет использования неописанной переменной, согласование типов в операторе присваивания и т.п., поскольку для спецификации подобных контекстных свойств необходимо передавать информацию между произвольно далеко отстоящими частями предложения
- Например, язык $\{wcw \mid w \in \{0,1\}^*\}$, абстрагирующий проблему декларации переменных до их использования, не контекстно-свободный

Грамматики с рассеянным контекстом

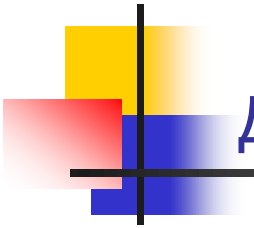
- Замена символа может зависеть от контекста, но нетерминал может быть заменен даже если контекст не смежен с ним
- **Определение.** Порождающей грамматикой с рассеянным контекстом называется четверка объектов G=(T, N, S, R), где:
  T – конечное непустое множество (терминальный словарь);
  N - конечное непустое множество (нетерминальный словарь);
  S – начальный символ - $S \in N$;
  R – конечное непустое множество правил (продукций), каждое из которых имеет вид
  $(A1, ..., An) \rightarrow (w1,...,wn)$, $n \geq 1$, *где* $Ai \in N$, и $wi \in (T \cup N)^+$
- Продукции грамматики с рассеянным контекстом - пары векторов, один из которых – заменяемые символы – это только нетерминалы, а другой – соответствующее количество непустых цепочек объединенного словаря
- Отношение '$\Rightarrow$' непосредственной выводимости на множестве цепочек:
  - Пусть $(A_1, ..., A_n) \rightarrow (w_1,...,w_n) \in R$, $x_i \in (T \cup N)^*$. Тогда:
    $x_1A_1 \ldots x_nA_nx_{n+1} \Rightarrow x_1\, w_1 \ldots x_n\, w_n\, x_{n+1}$.
  - Таким образом, сразу несколько нетерминалов, стоящих произвольно далеко в порождаемой цепочке, могут быть заменены одновременно на соответствующие им цепочки объединенного словаря
- ' $\Rightarrow^*$' означает рефлексивное транзитивное замыкание '$\Rightarrow$'.
- Язык, порождаемый грамматикой G, это множество $L(G)=\{w \in T^* \mid S \Rightarrow^* w\}$



Пример

- Грамматика с рассеянным контекстом, порождающая язык
  $L=\{a^n b^n c^n | n > 0\}$
- Такая грамматика будет иметь три продукции:
  $S \rightarrow ABC$
  $(A, B, C) \rightarrow (aA, bB, cC)$
  $(A, B, C) \rightarrow (a, b, c)$
- Вывод цепочки $a^3b^3c^3$:

$S \Rightarrow ABC \Rightarrow aAbBcC \Rightarrow aaAbbBccC \Rightarrow aaabbbccc$

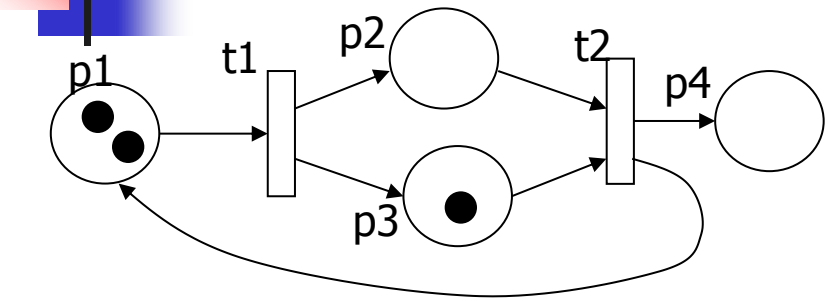


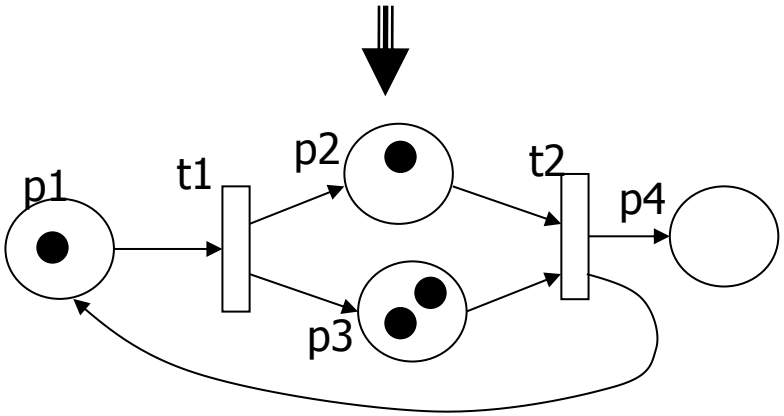
Другие модели порождения языков

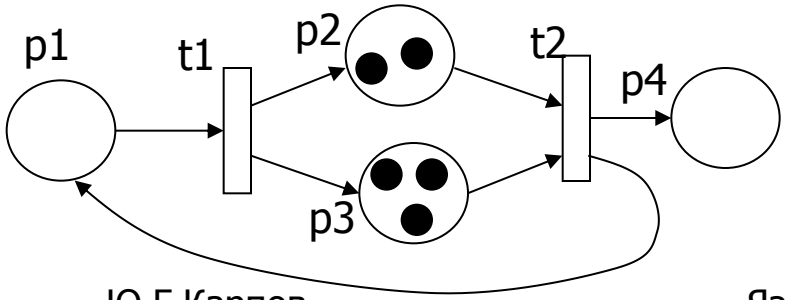
Грамматики Хомского являются моделью, которая лежит в основе подавляющего большинства идей в области спецификации формальных языков

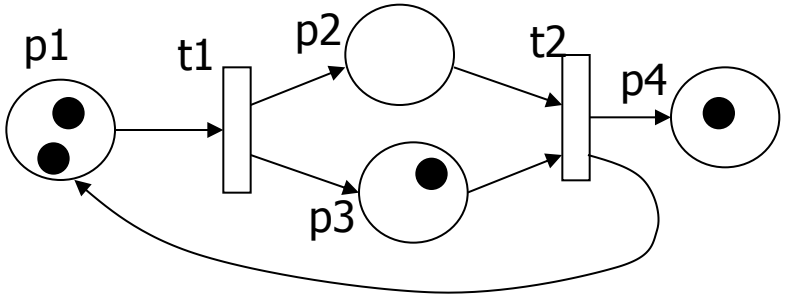
- В частности, нотация Бэкуса-Наура является просто более удобным представлением правил грамматик Хомского типа 2, синтаксические диаграммы – их наглядным графическим представлением, а грамматики с рассеянным контекстом разрешают векторную замену сразу нескольких нетерминалов в порождаемой строке, что является простым обобщением КС-грамматик.
- Но грамматики Хомского отнюдь не единственно возможный способ задания языков. Фактически, порождающим механизмом при задании языка может служить формальная модель любой природы, каким - то образом определяющая множество последовательностей объектов (символов) над конечным словарем

Пример сети Петри





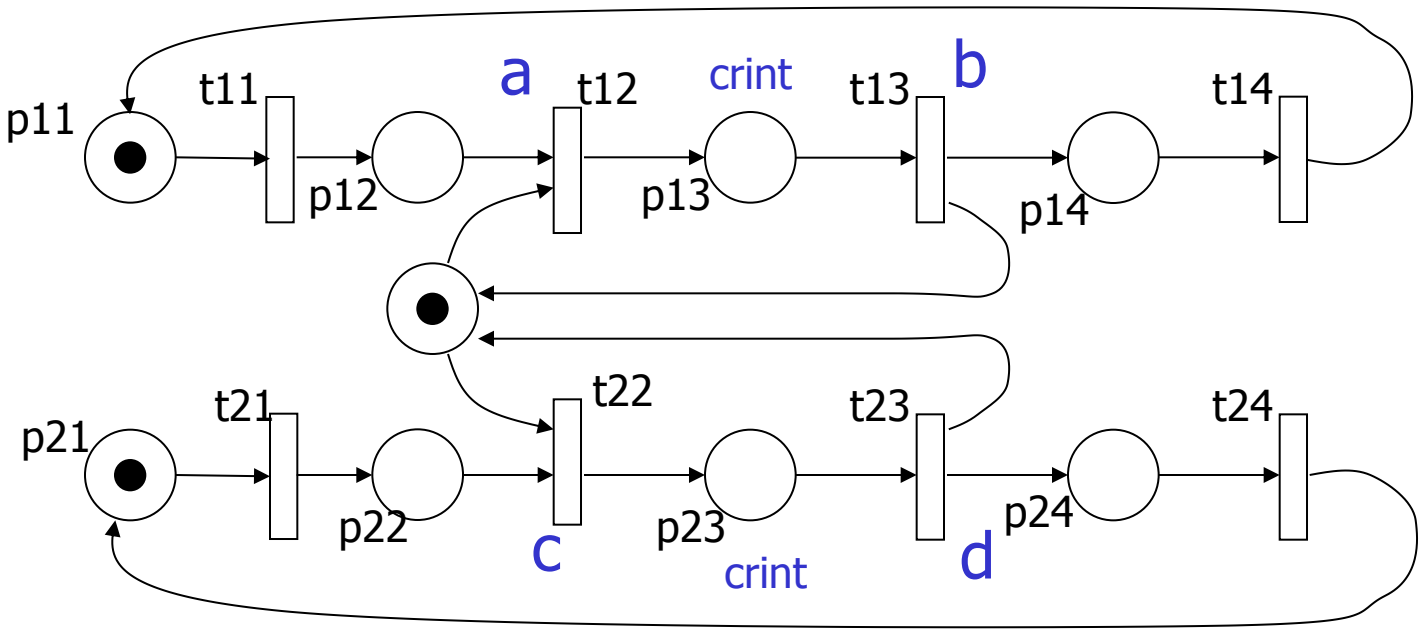




Пометим некоторые (существенные для конкретного приложения) переходы сети Петри символами конечного алфавита. Множество всех последовательностей пометок срабатываний переходов сети является языком, который называется *свободным префиксным языком сети Петри.* Таким образом, динамику сети Петри может определить язык

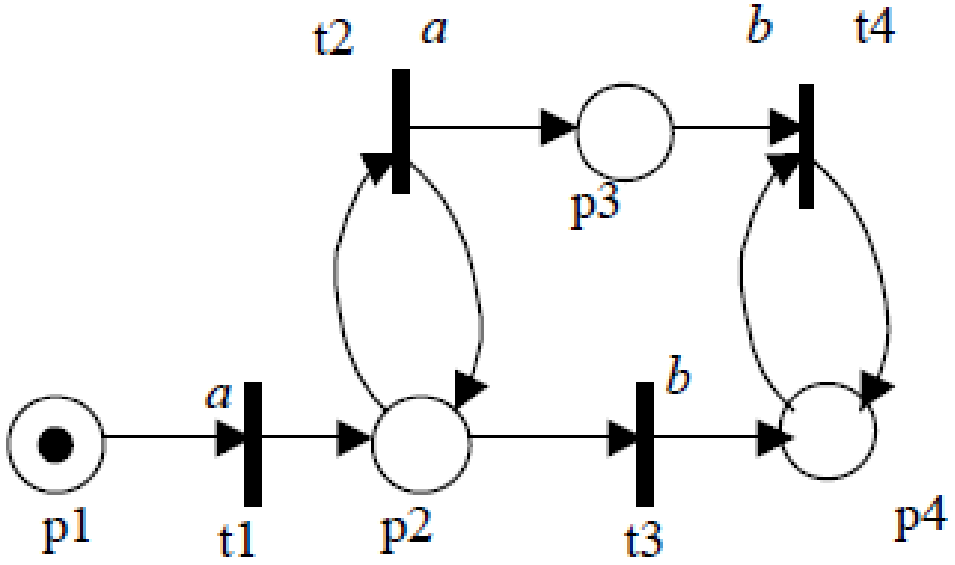
Свойства сетей Петри

Пример: синхронизация доступа параллельных процессов к общему ресурсу



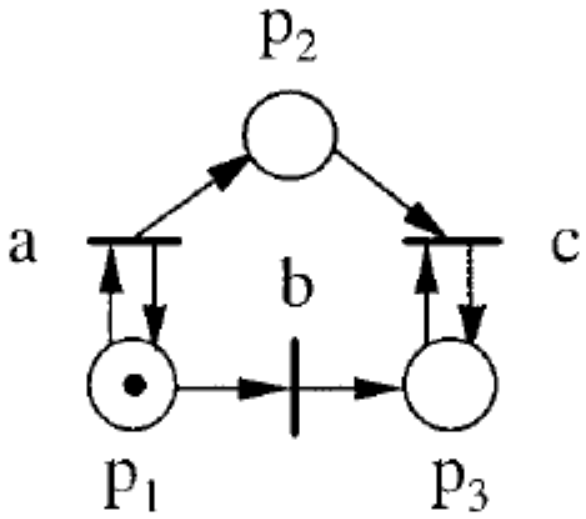
Интересующие нас события входа в критический интервал и выхода из него помечены. Язык, порождаемый этой сетью Петри, определяется регулярным выражением (ab+cd)*. Простейший анализ этого языка показывает, что проблема взаимного исключения здесь решена корректно

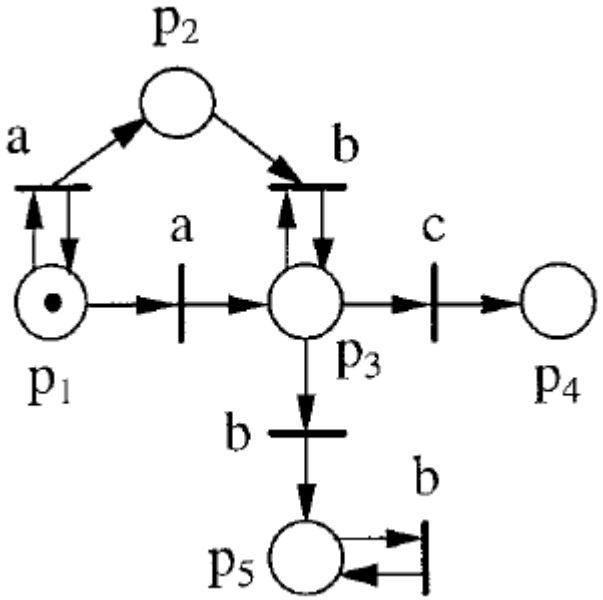
Пример: сеть Петри, порождающая язык $L=\{a^n b^n \mid n>0\}$



Сетью Петри для достижения финальной маркировки (0,0,0,1) порождается терминальный язык $L=\{a^n b^n \mid n>0\}$, который является КС-языком

Задачи





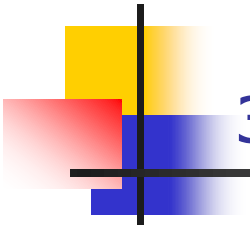
- Какие языки порождают сети Петри?

Порождающая мощность сетей Петри

Существуют КС-языки, которые не могут быть представлены сетями Петри. Например, это доказано для языка $\{wcw^R \mid w \in \{a, b\}^*\}$, w^R– зеркальное отображение w}. В то же время, существует сеть Пети, порождающая язык $\{a^n b^n c^n \mid N>0\}$, который не является КС-языком. Отсюда следует, что сети Петри несравнимы по порождающей мощности с магазинными автоматами.

Невозможность с помощью сетей Петри представить некоторые языки показывает, что они строго менее мощны, чем универсальный автомат – машина Тьюринга

Теорема. Класс помеченных сетей Петри строго мощнее класса конечных автоматов, несравним с классом магазинных автоматов и строго менее мощен, чем класс машин Тьюринга.

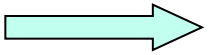


Заключение

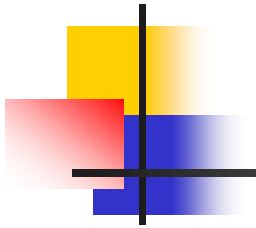
- Язык типа 0 может быть задан конечным образом недетерминированной машиной Тьюринга. Однако, алгоритма распознавания для любого языка типа 0 не существует
- Язык типа 1 может быть задан линейно-ограниченным автоматом. Существует общий алгоритм распознавания таких языков
- Любой КС-язык может быть распознан автоматом с магазинной памятью: по КС-грамматике Хомского такой МП-автомат легко строится. В общем случае такой МП-автомат – недетерминированный. Понятие МП-автомата впервые сформулировано Н.Хомским в 1962 г.
- По А-грамматике легко строится НДКА, по нему – эквивалентный ДКА, а по нему легко выполнить распознавание цепочки и, следовательно, трансляцию
- Автоматные грамматики Хомского (грамматики типа 3) порождают в точности языки, которые распознаются конечными автоматами
- Существуют другие модели задания языков: БНФ-нотация, синтаксические диаграммы, грамматики с рассеянным контекстом, сети Петри и другие
- БНФ-нотация и синтаксические диаграммы задают в точности КС-языки

Требования к знаниям студента по разделам курса

- Конечные автоматы
- Грамматики Хомского
- Атрибутная семантика
- Синтаксический анализ
- Трансляция ЯВУ



- иметь ясное представление понятий: формальный язык, грамматика, синтаксис, семантика
- понимать проблему двусмысленности фраз языков и идеи Хомского о синтаксически-ориентированном подходе к трансляции
- понимать идеи порождающих грамматик Хомского, того, как ‘работает’ порождающая грамматика
- знать классификацию грамматик, распознающих автоматов и порождающих грамматик
- уметь строить конечный автомат по автоматной грамматике, МП-автомат по КС-грамматике
- уметь использовать лемму о накачке для определения неавтоматности языков
- уметь упрощать и преобразовывать КС-грамматику к нормальным формам Хомского и Грейбах
- уметь выполнять левый и правый канонический вывод цепочки в КС-грамматике
- знать примеры порождающих грамматик других типов (сети Петри)
- уметь строить различные деревья вывода для цепочек, порождаемых неоднозначными КС-грамматиками



Спасибо за внимание