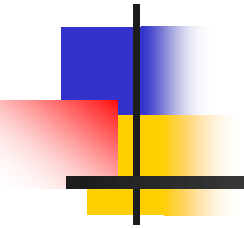


Автоматы и формальные языки

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

karpov@dcn.infos.ru

Курс по автоматам и формальным языкам

- Цель курса – изучение теоретических основ и принципов построения алгоритмов трансляции

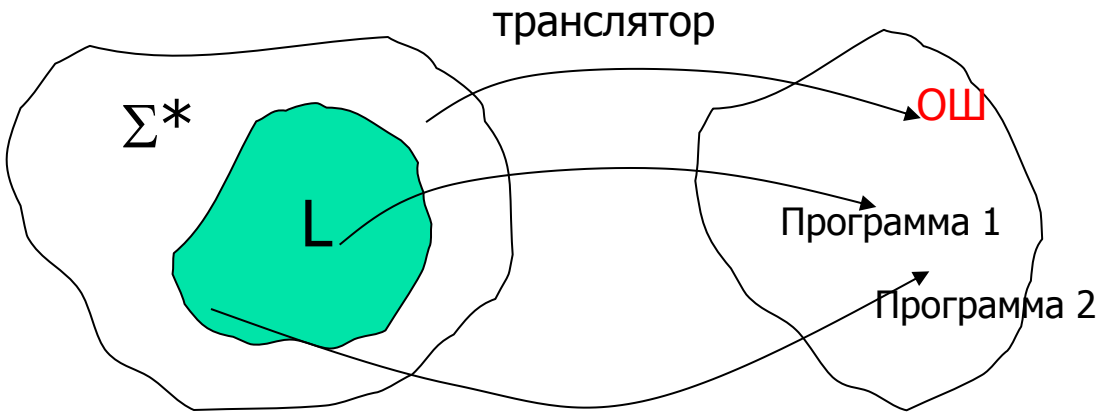
- Входные последовательности символов (цепочки) конечные, они состоят из элементов конечных множеств
- Эти множества называются словарями, а элементы множеств – символами
- Конечные последовательности символов называются цепочками (словами, предложениями, текстами)
- Основами трансляции являются теория и методы формального синтаксического и семантического анализа языков (в том числе языков программирования), а также алгоритмы анализа, основанные на этих теориях

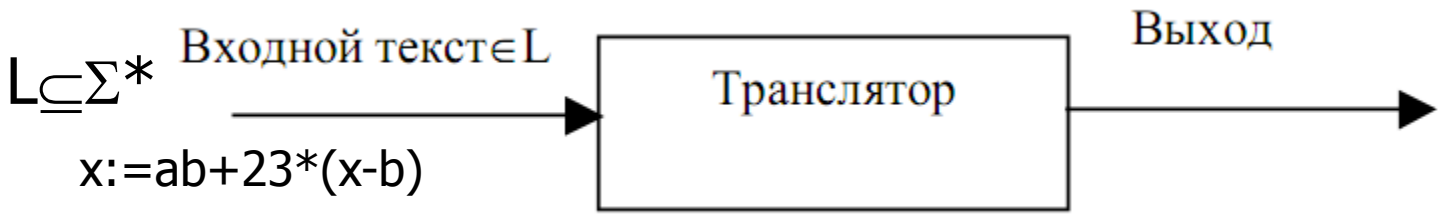
Трансляторы, формальные языки, автоматы

Σ- конечный словарь

Σ* - множество всех конечных цепочек из символов Σ

L – подмножество Σ*, язык





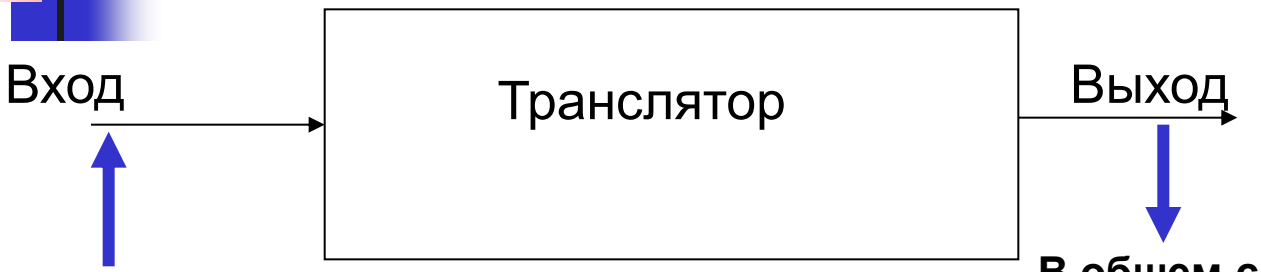
алгоритм обработки цепочек символов

программа

```
0: LOAD    15
1: PUSH   ‘23’
2: LOAD    21
3: LOAD    25
4: SUB
5: MULT
6: ADD
7: STORE   21
```

Поскольку абстрактные автоматы – это модели, использующиеся для обработки цепочек символов, то курс связан и с абстрактными автоматами

Цель курса



Цепочка
(конечная последовательность) символов

- программа на ЯВУ
- цепочка команд языка управления ОС
- фраза естественного языка
- описание ситуации
- описание действий по вязания кофты

В общем случае – что угодно

- команды стековой машины
- цепочка (предложение) другого языка;
- последовательность запрашиваемых действий;
- образ, картинка, граф-схема программы;
- команды вязальной машины для вязания кофточки;
- команды на исполнительные механизмы робота

Входы разные, выходы разные, но ИДЕИ и методы,
ЛЕЖАЩИЕ В ОСНОВЕ АЛГОРИТМОВ ТРАНСЛЯЦИИ – ОДНИ И ТЕ ЖЕ
Наша задача – изучить эти идеи и методы

Зачем изучать теорию формальных языков

- Трансляторы языков программирования высокого уровня пишутся не очень часто, но знание основ построения трансляторов необходимо каждому
- В любой области знаний нужен свой специализированный язык. Например, нужно проверить правильность теоремы: *"Даны три утверждения, истинных или ложных. Если из истинности первого следует истинность третьего, то верно, что если из истинности второго следует истинность третьего, то истинность первого или второго утверждения влечет истинность третьего".* Как провести ее доказательство?

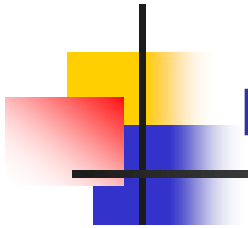
  Эта теорема на формальном ЯЗЫКЕ логики:

  $$(p \Rightarrow r) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \vee q \Rightarrow r))$$

  доказательство того, что эта формула – тавтология – тривиально

Любой интерфейс человека с машиной и межмашинный интерфейс - не аморфная последовательность 0 и 1. Последовательность имеет структуру, следовательно интерфейсы, входные языки пакетов программ и т.п. – все основаны на специализированных языках

С необходимостью разработки специализированных языков и построения транслятора для них сталкивается любой непримитивный программист



Важный, интересный материал

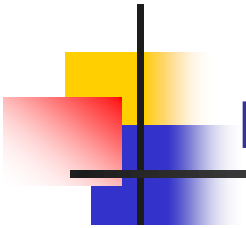
Специализированные формальные языки используются во всех областях деятельности человека, их изобретают для описания предметных областей, задания алгоритмов, …

"Границы моего языка есть границы моего мира"

Людвиг Витгенштейн

"Теория формальных языков – это цветок информатики, лепестками которого являются теория языков программирования и компиляция"

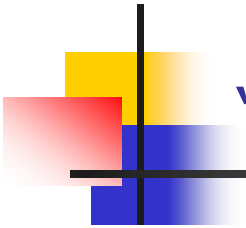
Альфред Ахо



Мотивация

- Мы будем изучать теорию –
  Теорию абстрактных автоматов и формальных языков –
  но с определенной практической целью: изучение методов построения трансляторов

- Изучение теории дает концепции и принципы, которые позволяют строить трансляторы и преобразователи любых языков (текстов)
- Теория формальных языков, формальных грамматик и синтаксический анализ - одна из наиболее полно исследованных областей Computer Science. Ее теоретические результаты решают практические задачи: дают базу для построения трансляторов любых языков
- Семантический анализ и оптимизация кода – два направления, в которых исследования продолжаются. В области синтаксиса ПОЧТИ все вопросы глубоко исследованы и хорошо поняты



“Деятельная” компонента

Курс по программной теории и инженерии не может быть чисто теоретическим: он должен включать обязательную часть

“learning by doing”

Каждый студент должен построить свой транслятор

Как это сделать?

На лекциях подробно разбирается компилятор, построенный для базового языка Milan (Mini Language)

Каждому студенту предоставляется его собственное расширение возможностей этого языка, например, введение дополнительных управляющих конструкций или типов

Наряду с расширением базового компилятора реализацией этих конструкций, от студента требуется полное, доскональное понимание работы базового компилятора

Курсовая работа

ЕСТЬ

будем вместе разрабатывать

Вход – программа на базовом языке MiLan

Базовый компилятор для языка MiLan

Выход – программа для виртуальной стековой машины

Разрабатывает студент

Вход – программа на модифицированном языке MiLan++

У каждого студента свое расширение языка MiLan

Компилятор для языка MiLan++

Выход – программа для виртуальной стековой машины (возможно, с доп. командами)

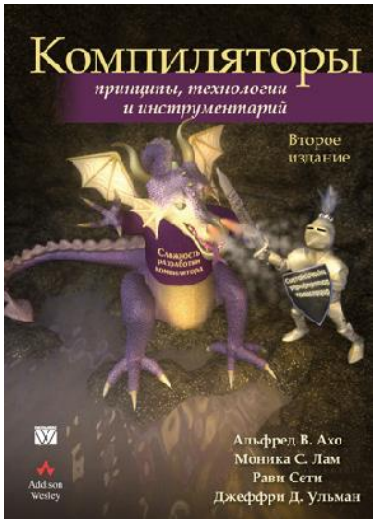
Организационные вопросы: отличия от предыдущего курса!!

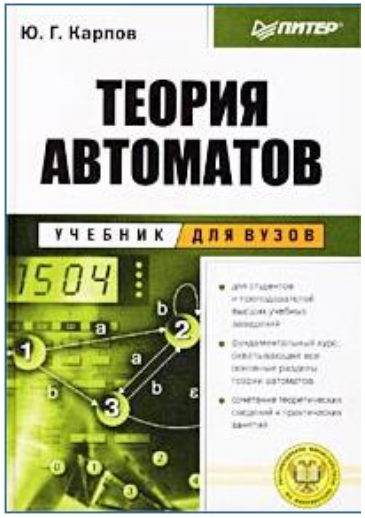
- Лекции
  - Каждая посещенная лекция дает +1 балл; посещение всех лекций дает +5 баллов
- Практическая работа
  - Практические занятия: посещение, активность и выполнение 10 самостоятельных работ от -10 до +15 баллов
  - Курсовая работа: построение своего расширения компилятора рекурсивного спуска базового языка Милан от - 10 до +5 баллов
  - Доп: за освоение Lex, Jacc (Bison), разработку компилятора сложных конструкций языков - до +10 баллов
  - Доп: за освоение библиотеки обработки регулярных выражений - до +10 баллов
- Экзамен письменный. Задачи на сайте //dcn.ftk.spbstu.ru/~karpov
  - за решение всех задач на экзамене до **80** баллов
- Всего можно набрать 140 баллов (15лек + 5лек + 15практ + 25курс + 80экз)
- Общая оценка выводится по сумме набранных баллов:

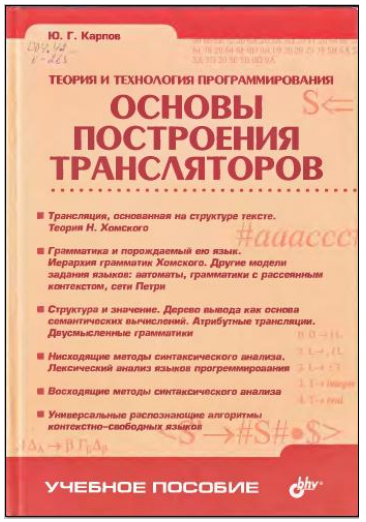
Σ баллов	0 - 19	оценка	1
	20 - 49		2
	50 - 75		3
	75 - 99		4
	100 и более		**5**

Автоматы и языки - в рамках одной теории

- Теория формальных грамматик и формальных языков тесно связана с теорией автоматов







М.В.Мозговой. Классика программирования: Алгоритмы, Языки, Автоматы, Компиляторы

Д. Хопкрофт, Р. Мотвани, Д. Ульман. Введение в теорию автоматов, языков и вычислений

А.Ахо, Р.Сети, Д.Ульман. Компиляторы: принципы, технологии и инструменты

Ю.Г.Карпов. Теория автоматов. Питер, СПб, 2003

Ю.Г.Карпов. Теория и технология программирования. Основы построения трансляторов. БХВ, 2006 (есть в библиотеке и на кафедре)

Т.Пратт, М.Зелковиц. Языки программирования: разработка и реализация. : Питер, 2002

Структура курса

- Автоматы и языки – 4 л
- Синтаксис: порождающие грамматики Хомского – 3 л
- Семантика: атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Трансляция контекстно свободных (КС) языков – 6 л
- Доп. лекции - 2 л

========

17 л

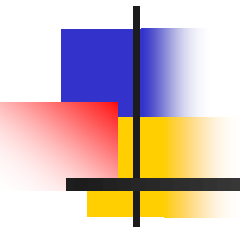
- Что должен знать и уметь студент (разбор задач по курсу) 1 л (если успеем)

========

18 л

Структура курса

- Автоматы и языки – 4 л
  - Лекция 1. Формальные языки. Примеры языков. Грамматики. КА
  - Лекция 2. Теория конечных автоматов-распознавателей
  - Лекция 3. Трансляция автоматных языков
  - Лекция 4. Регулярные множества и регулярные выражения
- Порождающие грамматики Хомского – 3 л
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
- Дополнительные лекции - 2 л



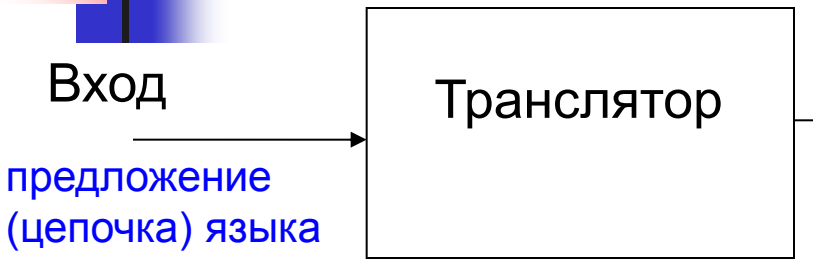
Лекция 1

Формальные языки
Примеры языков
Грамматики
Конечные автоматы как грамматики

Компиляторы, интерпретаторы, препроцессоры

- Входной язык называется *исходным* (source language), а входная программа на этом языке называется *исходным кодом* (source code)
- Машинный язык, на который осуществляется трансляция, называется *объектным языком,* результат трансляции называется *объектным кодом*
- Компьютер, на котором оттранслированная программа будет запущена, называется *целевой машиной (target machine),* он не обязательно совпадает с типом машины, на которой работает транслятор
- Если задачей является трансляция в язык низкого уровня, например, машинный язык или язык ассемблера, то такой транслятор называется *компилятором*
- Другая возможная задача при трансляции программы на алгоритмическом языке – непосредственное выполнение транслятором операторов исходного кода. В этом случае транслятор называется *интерпретатором*
- *Языковой процессор* – программа, переводящая текст во что-нибудь, в программу на другом языке или, например, в картинку

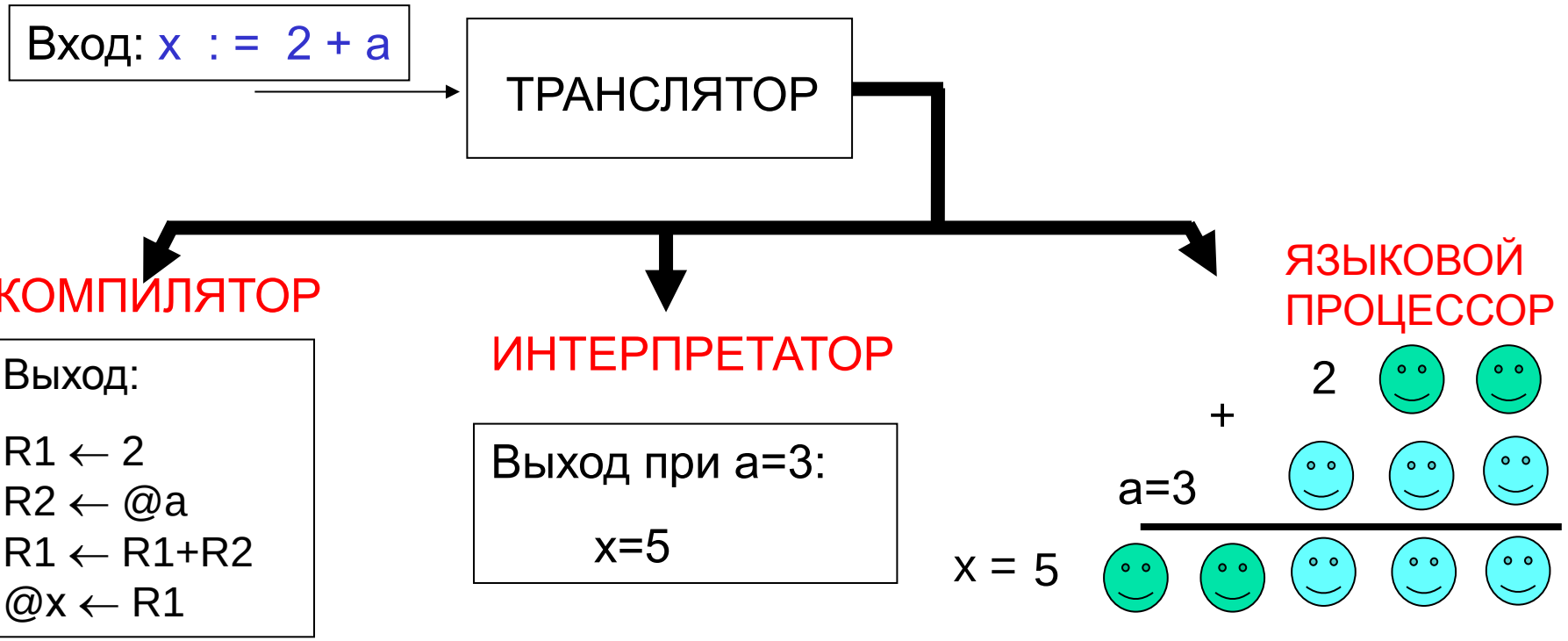
Трансляторы, компиляторы, интерпретаторы, ...



КОМПИЛЯТОР

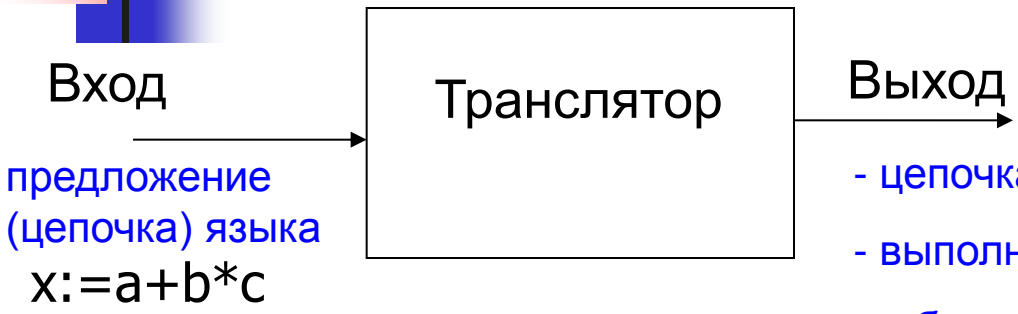
ИНТЕРПРЕТАТОР

ЯЗЫКОВОЙ ПРОЦЕССОР



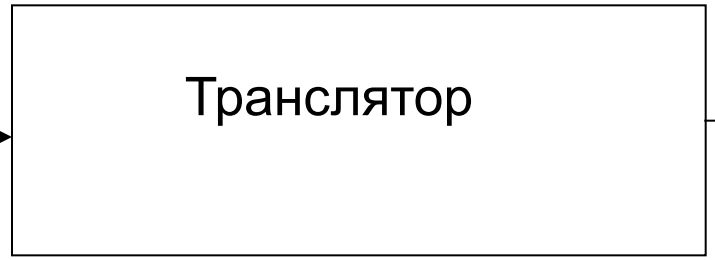
Для получения любого выхода нужна теория построения трансляторов

Отличия интерпретатора от компилятора



- Компилятор при обработке оператора присваивания x:=a+b*c построит и выдаст последовательность команд объектного языка, осуществляющую вычисление значения выражения a+b*c и засылку его в область памяти, отведенную для x
- Интерпретатор непосредственно выполнит действия, предписываемые этим оператором присваивания, т.е. выполнит a+b*c для текущих значений правой части и перешлет полученное значение в область памяти для x. Классическим языком, для которого разработаны интерпретаторы, является Бэйсик
- Языковой процессор может цепочку x:=a+b*c перевести
  - в картинку
  - в оператор присваивания на C: пару символов ':=' заменит на '=='
  - ...

Пример транслятора - интерпретатора



Запись решения шахматного этюда в стандартной шахматной нотации

1. Kpd2 Kb2
2. Kpc2 Kpa5
3. Kpb3 Kpa5
4. Kd6! c4+
5. K:c4+ Kpb5
6. ... ...

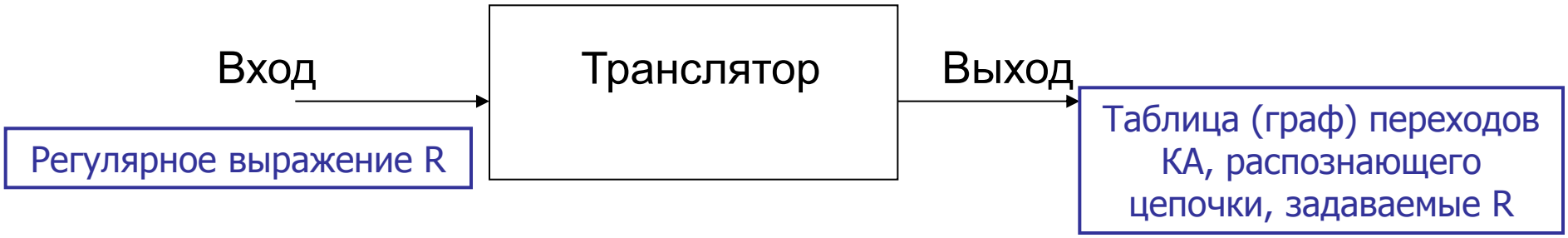
Здесь транслятор – интерпретатор, для каждого хода выдает:
с какого поля на какое поле нужно переместить фигуру

Движение фигур на шахматной доске

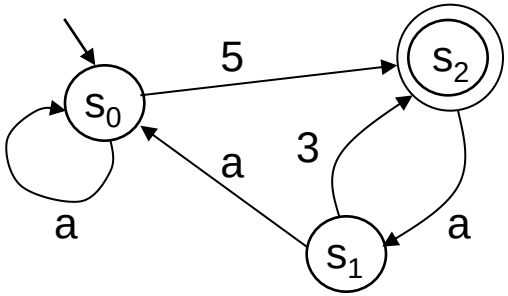
- Транслятор – это преобразователь информации, который по входной цепочке символов выдает нужный выход

- Предложение (слово, цепочка) языка всегда конечно: предложение есть соединение слов, выражающее законченную мысль

Пример языкового процессора



R=/([^\s]+)\d+[^\s \.]+/

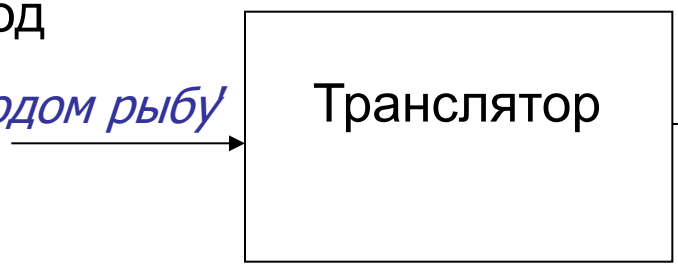


Пример языкового процессора

предложение (цепочка) языка

Вход

`*старик ловил неводом рыбу*'



Выход

- Языковый процессор может транслировать текст в образы

Основная цель курса: научить создавать трансляторы

Вопрос:

Как создать свой транслятор – компилятор или интерпретатор, и вообще какую-нибудь программу для обработки некоего текста, представляющего любую информацию:

- программу на некотором новом языке программирования,
- входной файл языка управления работами,
- просто какие-нибудь данные, вводимые с клавиатуры?

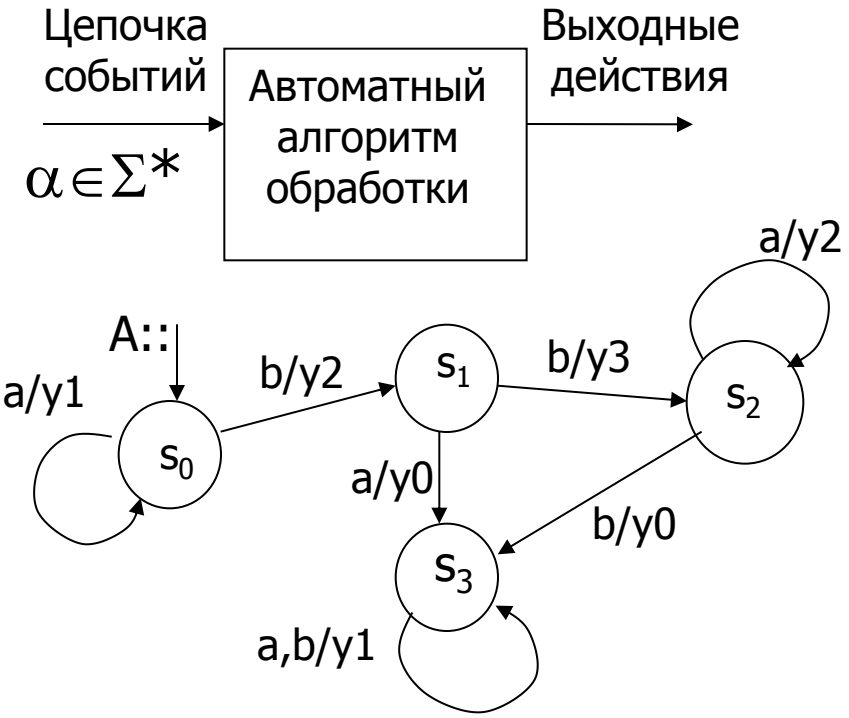
Ответ:

Теория формальных языков – это наука о том, как разобрать входную информацию, вычисляя ее “смысл”.

Создать свой анализатор, свой парсер и “генератор” выходных данных можно на основе теории, представленной в курсе лекций

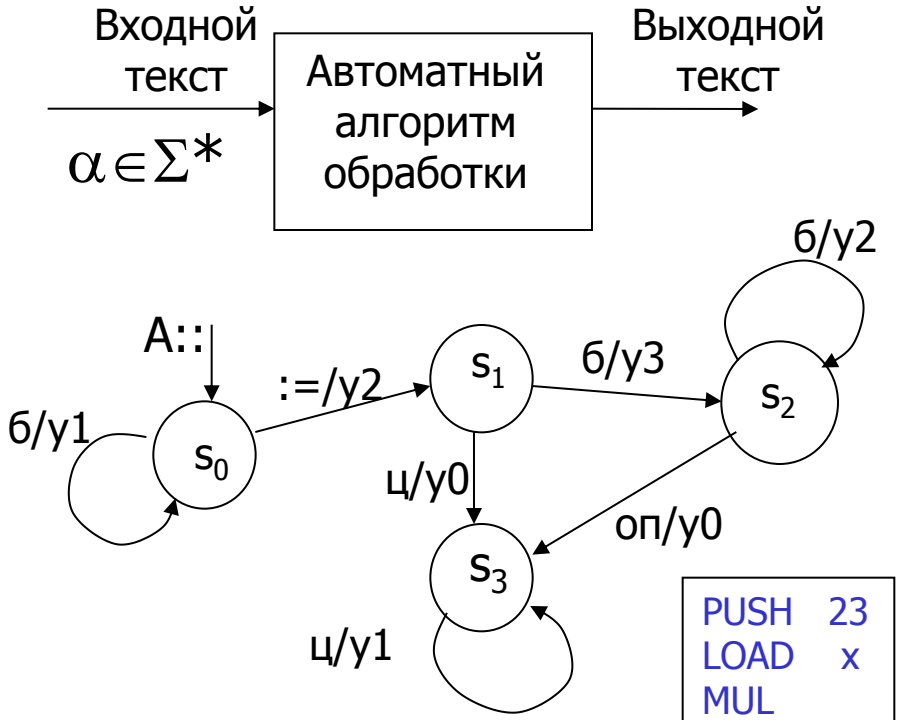
Связь теории автоматов и теории языков

Классический конечный автомат



aabab $\Rightarrow$ $y_1y_1y_2y_0y_1$

Конечный автомат как "обработчик" предложений языков



z:=23*x-b $\Rightarrow$

```
PUSH   23
LOAD   x
MUL
LOAD   b
SUB
STORE  z
```

Определение и обработка языков - это одна из самых неожиданных областей применения автоматов (не обязательно только конечных)

Языки – что это? Как определить язык формально?

- Неформально
  - *Язык* — система знаков и правила их соединения для передачи информации последовательностями знаков
- Формальное определение языка должно быть очень широким и абстрактным, оно должно покрывать:
  - языки естественные “Мама мыла раму”
  - языки программирования “begin int k = 5; print (2*k+8) end”
  - язык Римских чисел “CMXCII” (992), “MMXI” (2011)
  - язык запросов к БД
  - язык формул логики высказываний “$a \vee (b \oplus c),\ b \Rightarrow (a \downarrow c \vee d)$”
  - язык азбуки Морзе “ · · — — · · ” – буква э, “— — — —” – буква м,
  - язык для станков с ЧПУ (операторы перемещения инструмента, резать по дуге, поднять инструмент, подача в точку, резать от и до, ...)
  - язык управления ОС
  - язык для описания алгоритма вязания кофточки
  - и т.д.

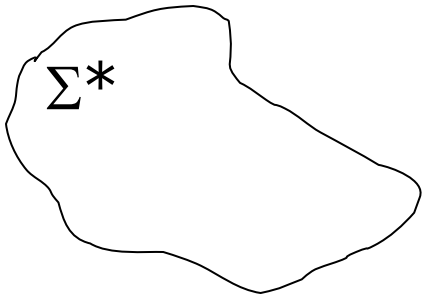
Основные определения и понятия

- Конечное множество объектов называется словарем.
- Элементы словаря называются символами.
- Цепочкой (символов) над словарем называют произвольную конечную последовательность символов словаря.
- **Пример.** Пусть Σ – словарь, содержащий три символа, *a,b,c.* Над словарем Σ можно построить различные цепочки, например, α=*abbca,* β=*cca,* γ=*b.*
- Важна пустая цепочка, для ее обозначения введен специальный символ ε.

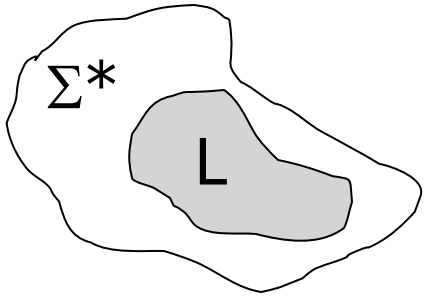
Σ={a, b, c}

Σ^*= {ε, a, b, c, aa, ab, abbca, ...}

L = {abb, abcbb, bbcc, aa, ab, ...}



$L \subseteq \Sigma^*$



- Самое общее определение языка: Формальный язык – это некоторое множество предложений (цепочек), построенных из символов некоторого конечного словаря

Операции над цепочками

- Цепочки строятся из символов конечного словаря
- Пусть Σ – словарь, содержащий три символа: a, b, c. Σ={a, b, c}
  - над Σ можно построить различные цепочки, например, α=abbca, β=cca
- Операция конкатенации двух цепочек: $\alpha\beta$=abbcacca; $\beta\alpha$=ccaabbca
- Пустая цепочка ε – единица конкатенации: $\varepsilon\alpha = \alpha\varepsilon = \alpha$
- Пусть $\alpha\beta\gamma$ - цепочка над каким-нибудь словарем. α,β,γ-тоже цепочки

  α - префикс цепочки $\alpha\beta\gamma$

  β - подцепочка цепочки $\alpha\beta\gamma$

  γ - суффикс цепочки $\alpha\beta\gamma$

  В любой цепочке можно выделить (неоднозначно!) и префикс, и суффикс
- Операция подстановки в цепочку вместо одной подцепочки другой цепочки
  - Пусть правило замены: cac $\rightarrow$ **b**

    Цепочку abbcacca по этому правилу замены можно преобразовать в цепочку abb**b**ca.

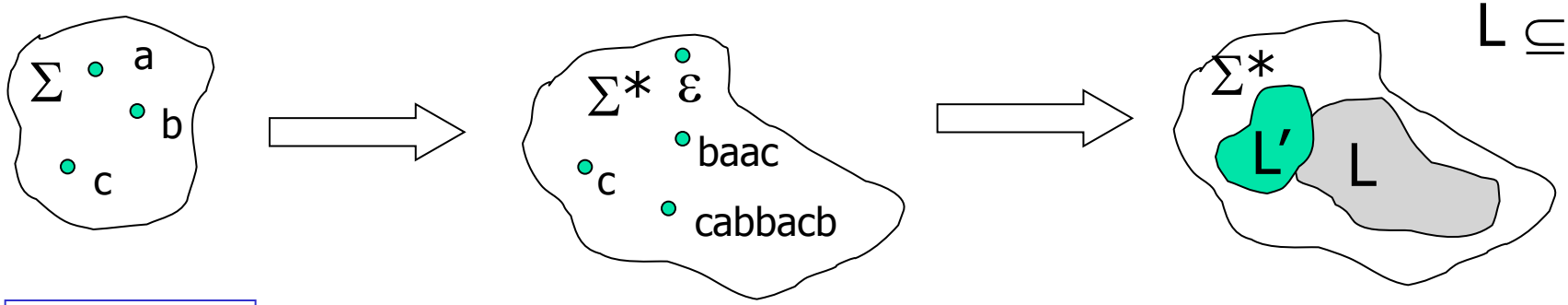
    Это можно записать так: abbcacca $\Rightarrow$ abb**b**ca

Язык: формальное определение

Языком над конечным словарем Σ называется произвольное множество конечных цепочек над этим словарем

- Цепочки языка называются словами (иногда предложениями)
- Над конечным непустым словарем можно определить бесконечное количество слов конечной длины (счетное множество)
- Очевидно, что над конечным непустым словарем можно определить бесконечное количество языков, т.е. подмножеств множества всех возможных слов (континуум, т.к. число подмножеств счетного множества имеет мощность континуум)

$\Sigma=\{a, b, c\}$ $\Sigma^*= \{\varepsilon, a, b, c, aa, ab, abbca, ...\}$ Язык – подмножество Σ^*

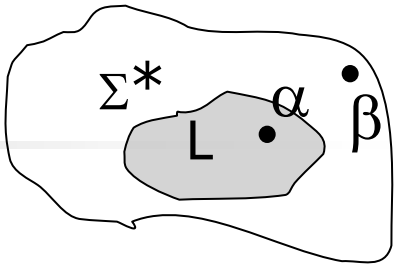


Словарь конечен

Число слов над конечным словарем бесконечно (счетно)

Число языков над конечным словарем континуум

Примеры языков



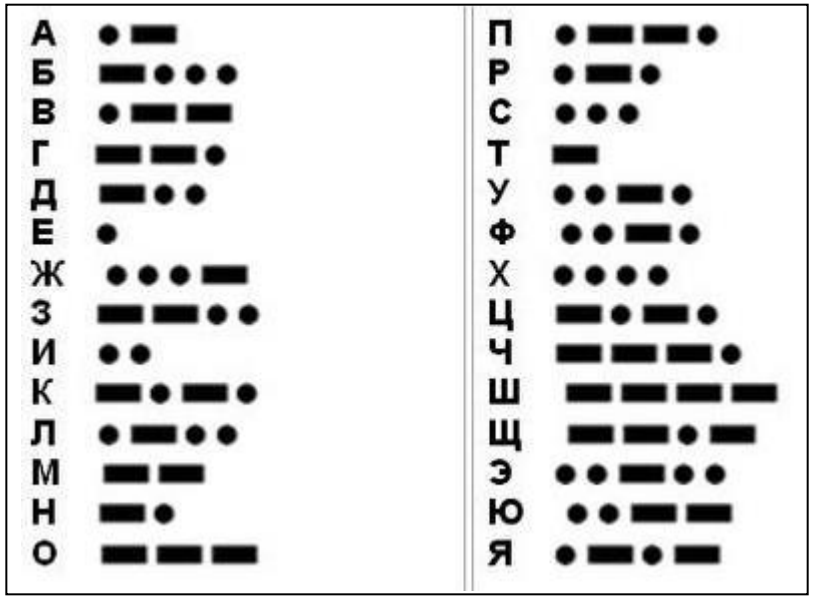
- Σ_1={a,b,c} L_1={abc, cc} – конечный язык
  cc$\in L_1$ cbc$\notin L_1$
- Σ_2={a,b,c} L_2= $\varnothing$ - пустой язык
  cc$\notin L_2$
- Σ_3={a,b,c} L_3= Σ^* - все возможные цепочки из a, b и c – принадлежат L_3
  acc$\in L_3$, так же, как и любая цепочка из символов словаря Σ_3
- Σ_4={a,b,c} L_4={$a^n bc^m$| n, m≥0} – язык L_4 – не из всех цепочек!
  aaaabcc$\in L_4$ cbaa$\notin L_4$
- Σ_5={a,b,c} L_5={$a^n bc^n$| n≥0 }
  aabcc$\in L_5$ cbaa$\notin L_5$
- Σ_6={a,b,c} L_6={$\alpha \in \Sigma_6^*$ | в α количества вхождений a, b и c равны}
  ccbaba$\in L_6$, ccb$\notin L_6$
- Σ_7={0,1} L_7=множество четных двоичных чисел
  100$\in L_7$ 0111101$\notin L_7$

Пример языка: азбука Морзе

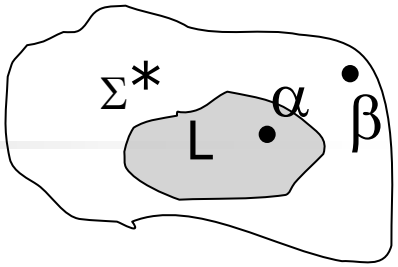
Словарь $\Sigma = \{ \bullet, - \}$

Язык азбуки Морзе – все кодировки букв (их конечное число)

Слова – последовательности букв – отделяются друг от друга разделителем (пробелом)



Примеры языков



- Σ_8={'+', '-', '0',..., '9'}; L_8= множество целых констант

  -67∈L_8 +234∈L_8 -3∈L_8 +2+3...4∉L_8

- Σ_9={ '(' , ')' } L_9=множество правильных скобочных выражений

  (( ))( )∈L_9))(( )∉L_9

- Σ_{10}={'+', '-', '0',...,'9', '.' } L_{10}= множество вещественных констант

  +6.7∈L_{10} 23+.+4..6∉L_{10}.

- Σ_{11}={a} L_{11}={$\alpha \in \Sigma^*$ | $|\alpha|$ = 2k, где k = 0, 1, ...}

  aaaa∈L_{11} a∉L_{11} (цепочки из четного числа a)

- Σ_{12}={a} L_{12}= цепочки из a длиной n^2 $\{a^{n^2} | n = 0,1,...,\}$

  aaaa∈L_{12} a∈L_{12}, aaa∉L_{12}

Языки могут быть конечными и бесконечными. Словарь всегда конечен

Примеры: Конечные языки: Азбука Морзе, L={abc, cc}, пустой язык, ...
Бесконечные языки: L_8, L_9, L_{10}, ...

Примеры языков (3)

- Σ_{13}={большой, мощный, тяжелый, …, кран, локомотив, контейнер, …, ставит, ведет, кладет, везет,…}; L_{13}= русский язык

большой локомотив везет тяжелый мощный кран $\in L_{13}$

молодая красивая студентка мчалась галопом в карете по пыльной дороге $\in L_{13}$

большой большой студентка контейнер красивый $\notin L_{13}$

- Σ_{14}={'+','-', …, '0',…, '9', …, begin, end, if, …, }; L_{14}=язык Паскаль

end begin if :: begin 0:== ; $\notin L_{14}$

begin int a end $\in L_{14}$

- Σ_{15}={'+', '-',… ,'0',…,'9', …, begin, end, …}; L_{15}= язык MiLan (Mini Language)

end begin if :: begin 0:== ; $\notin L_{15}$

```
begin
        a :=5;
        input (x);
        a :=x+a*3;
        while x>0 do a:=a*2; x:=x-1
  end
```

$\in L_{14}$

Язык Милан – основной язык для наших примеров, для него будет строиться компилятор в рамках курсовой работы

Операции над языками

- Поскольку языки – это множества цепочек, то все операции над множествами применимы и к языкам:

  $L_1=\{\varepsilon, acbac, b, ac\}$; $L_2=\{b, bca\}$;

  - Пример: теоретико-множественные операции

    $L_1 \cup L_2 = \{\varepsilon, acbac, b, ac, bca\}$; $L_1 \cap L_2 = \{b\}$; $L_2 \setminus L_1 = \{bca\}$

Специальные операции над языками

- Конкатенация языков: $L_1L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\}$; $L^{n+1} = L^nL$; $L^0=\{\varepsilon\}$
  - Пример: $L_1L_2 = \{b, bca, acbacb, acbacbca, bb, bbca, acb, acbca\}$
- Итерация (звезда Клини): $L^* = \bigcup_{k=0,1,\dots} L^k = \{\varepsilon\} \cup L \cup LL \cup LLL \cup LLLL \cup \dots$

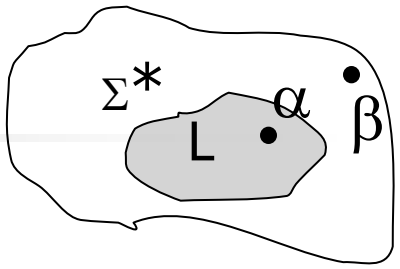
  $L^+ = \bigcup_{k=1,2,\dots} L^k = L \cup LL \cup LLL \cup LLLL \cup \dots$
  - Примеры: $L_2^* = \{\varepsilon, b, bca, bb, bbca, bcab, bcabca, bbbca, bbbca, \dots\}$

  $\Sigma = \{a, b, c\}$; $\Sigma^* = \{\varepsilon, a, b, c, ab, bb, abbca, \dots\}$ – все цепочки из a, b, c
- Префиксное замыкание языка: $Pref(L) = \{\alpha \mid (\exists \beta \in \Sigma^*): \alpha\beta \in L\}$
  - Примеры: $L_1=\{\varepsilon, acbac, b, ac\}$; $Pref(L_1) = \{\varepsilon, b, a, ac, acb, acba, acbac\}$ – дополнение языка префиксами

    $\Sigma = \{a, b, c\}$; Σ^*= все возможные цепочки. $Pref(\Sigma^*) = \Sigma^*$

    $L=\{\varepsilon, b, a, ac, bc, bca\}$, $Pref(L) = L$ префиксно-замкнутый язык

Как задать язык? Грамматики

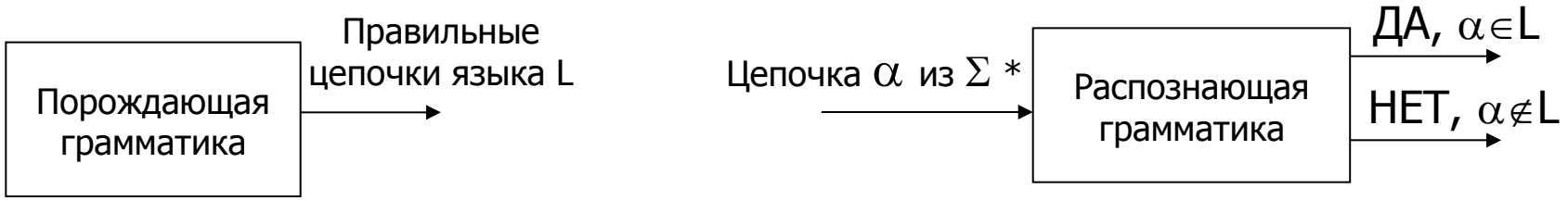


- Язык – в общем случае - бесконечное множество цепочек

  Но как его задать конечным образом, не перечисляя все цепочки?

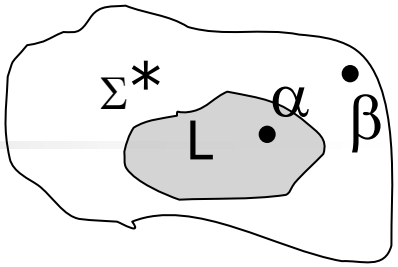
- Что такое L_9 – "*множество правильных скобочных выражений*"?
  Что такое "*скобочное выражение*"? Что такое "*правильное*"?
  Как задать новый язык программирования?

- *Любая конечная формальная модель для задания языка называется грамматикой*

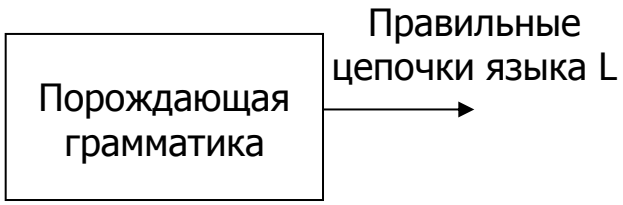


Формальная грамматика — способ выделения некоторого подмножества из множества всех возможных предложений.
Различают порождающие и распознающие грамматики;
первые задают правила, с помощью которых можно построить любое предложение языка,
вторые позволяют по данному предложению определить, входит ли оно в язык

Пример порождающей грамматики



- Правила порождения бесконечного множества цепочек, не перечисляя их все



Например, оракул (диктатор):

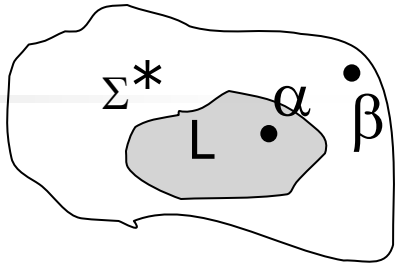
все, что он изрекает, правильно!

Например, партийная дисциплина:

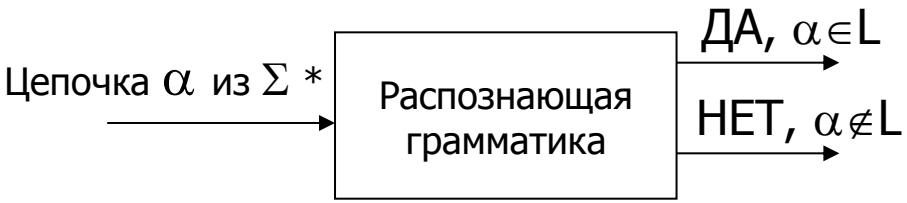
все, что наша партия заявляет, является правильным!

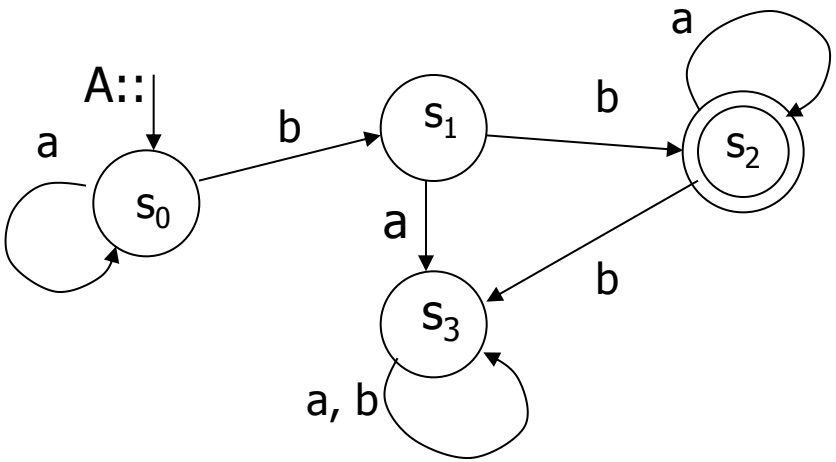
Мы вернемся к порождающим грамматикам позже

Распознающие грамматики



- Конечный набор правил, алгоритм, который по введенной цепочке определяет, принадлежит цепочка языку, или нет





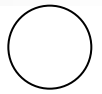
Конечный автомат без выхода можно считать распознающим механизмом, распознающей грамматикой,
КА может задавать язык!
Выходом автомата является тип состояния (Допускающее, НЕ Допускающее)

aaabba $\in L_A$
bbaa $\in L_A$
abb $\in L_A$

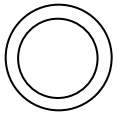
aaa $\notin L_A$
abab $\notin L_A$
baaaba $\notin L_A$
bbba $\notin L_A$

Конечный автомат-распознаватель

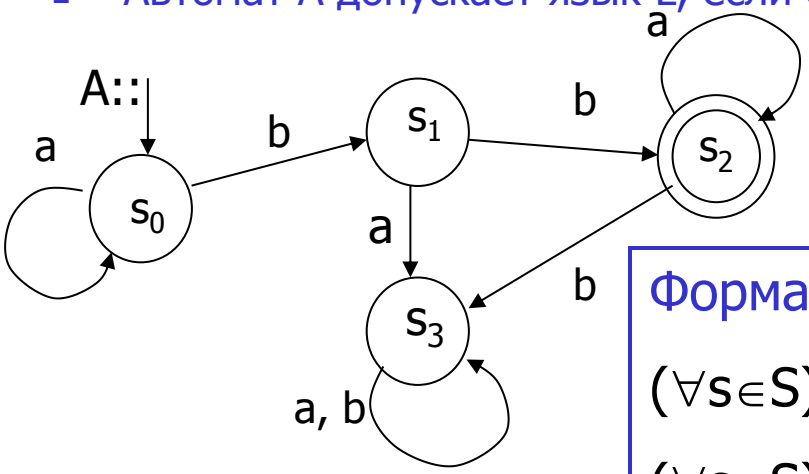
не допускающее состояние



Допускающее состояние



- Конечный автомат-распознаватель $A=(S, \Sigma, s_0, \delta, F)$, где:
  - S – конечное множество состояний
  - Σ – конечное множество входных символов
  - $s_0 \in S$ – начальное состояние
  - δ: $S \times \Sigma \to S$ – функция переходов
  - $F \subseteq S$ – множество финальных (допускающих) состояний
- Определения. Автомат A допускает (распознает) цепочку, если эта цепочка переводит A из начального в одно из финальных состояний
- Автомат A допускает язык L, если он допускает все цепочки этого языка, и только их



$$L_A = \{ \alpha \in \Sigma^* \mid \delta^*(s_0, \alpha) \in F \}$$

Формально: δ^*: $S \times \Sigma^* \to S$

$(\forall s \in S)\ \delta^*(s, \varepsilon) = s;$

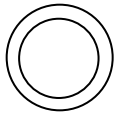
$(\forall s \in S)\ (\forall \alpha \in \Sigma^*)(\forall a \in \Sigma)\ \delta^*(s, \alpha a) = \delta(\delta^*(s, \alpha), a)$

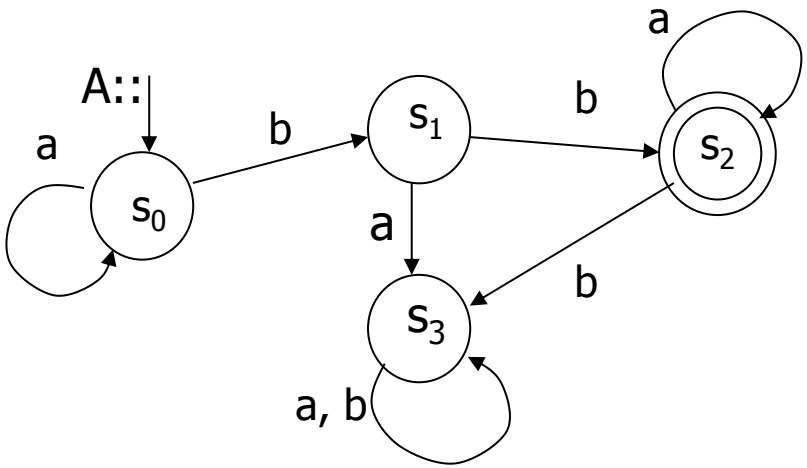
Пример конечного автомата-распознавателя

- Конечный автомат-распознаватель A=(S, Σ, s_0, δ, F), где:
  - S – конечное множество состояний
  - Σ – конечное множество входных символов
  - $s_0 \in S$ – начальное состояние
  - δ: $S \times \Sigma \to S$ – функция переходов
  - $F \subseteq S$ – множество финальных (допускающих) состояний

не допускающее состояние

Допускающее состояние





Формально: A=(S, Σ, s_0, δ, F)

S=(s_0, s_1, s_2, s_3); Σ = {a, b};

δ: $S \times \Sigma \to S$

$\delta(s_0, a)=s_0$; $\delta(s_0, b)=s_1$; $\delta(s_1, a)=s_3$;
$\delta(s_1, b)=s_2$; $\delta(s_2, a)=s_2$; $\delta(s_2, b)=s_3$;

F={s_2}.

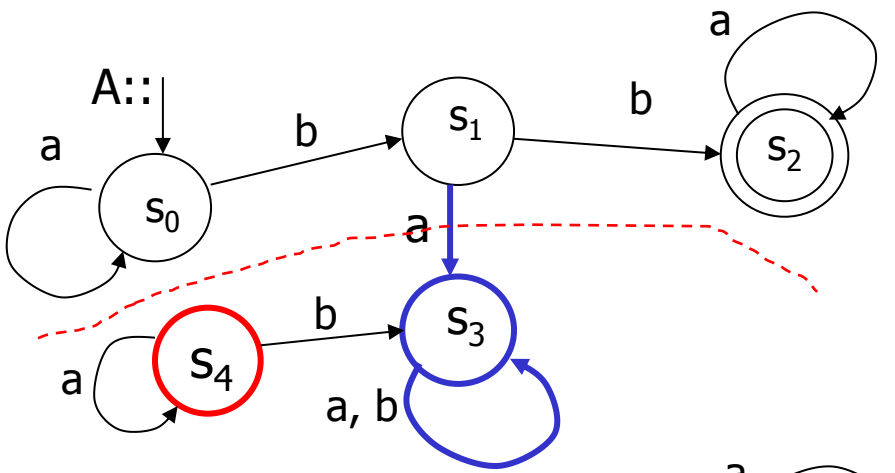
Конечный автомат-распознаватель

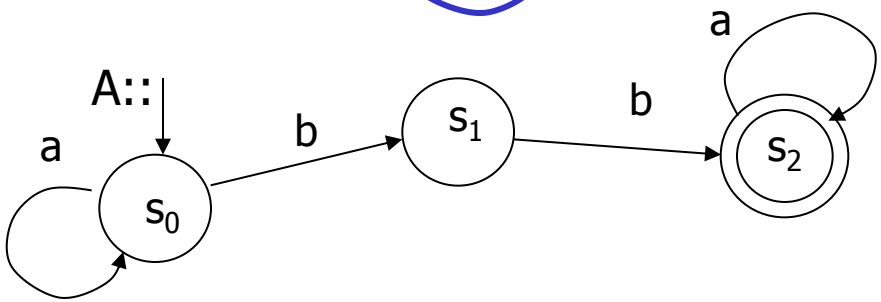
не допускающее состояние

Допускающее состояние

В автомате из состояния могут быть переходы не для всех входных символов

$L_A = \{a^n b\ b\ a^m \mid n, m \geq 0\}$





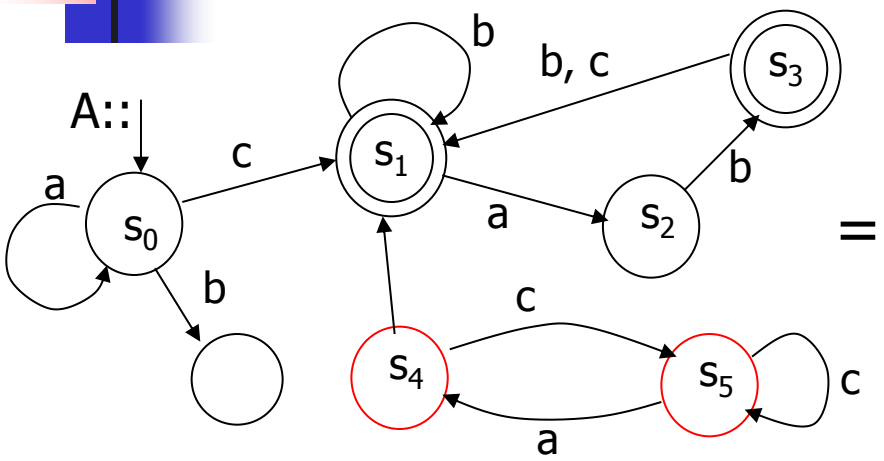
Состояние s4: автомат в него никогда не попадет

Состояние s3: если автомат в него попал, то никогда больше не попадет в допускающее состояние

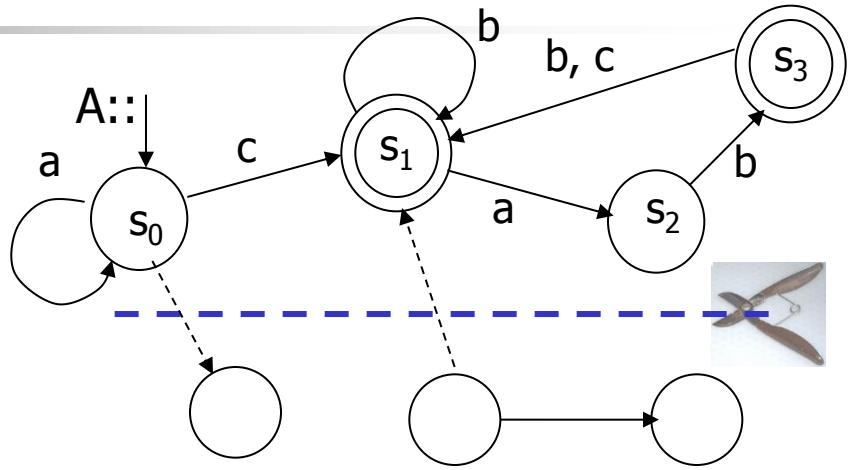
Состояние s_4 недостижимое
Состояние s_3 некодостижимое

Такие состояния и соответствующие пути часто на рисунке выбрасывают. Это упрощает визуальное восприятие и облегчает понимание функционирования КА. Но если они реально есть, то они означают наличие дедлока

Достижимые и недостижимые состояния



=



Достижимо(s) $\Leftrightarrow (\exists \alpha \in \Sigma^*)$: $\delta(s_0, \alpha)=s$

Недостижимо(s) $\Leftrightarrow (\forall \alpha \in \Sigma^*)$: $\delta(s_0,\alpha)\neq s$

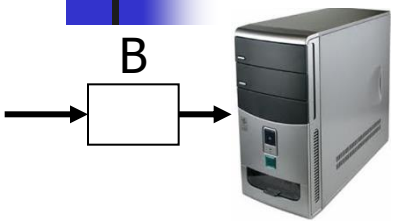
$\Sigma_A = \{ a, b, c \}$

- Общепринятая условность: рассматриваются только “приведенные” автоматы, у которых все состояния достижимы и кодостижимы. Выбрасывание недостижимых и некодостижимых состояний не изменяет языка, распознаваемого автоматом. Такая операция называется trimming (подстригание)

Задача 1: определите формально, какие состояния являются кодостижимыми

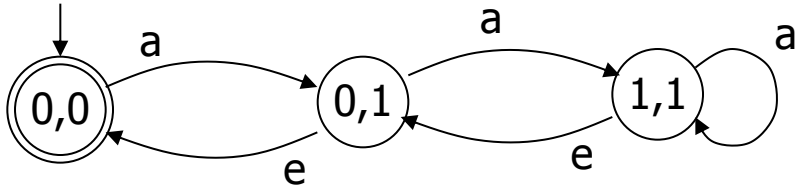
Задача 2: постройте алгоритм тримминга

Пример КА для дискретной событийной системы



- Процессор обрабатывает поток работ, имея на входе буфер, который может хранить не более одной работы. Если пришедшая работа нашла систему занятой, эта работа отбрасывается

a: событие прихода работы
e: событие завершения обработки
состояние: <буфер B, процессор>

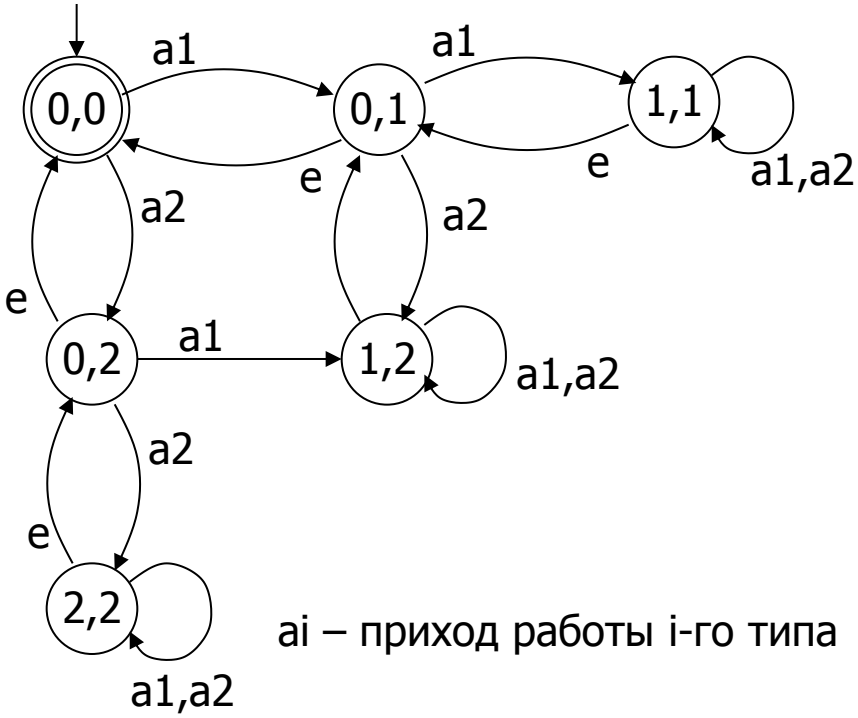


Усложнение. Работы во входном потоке могут быть двух приоритетов: 1 и 2. Приоритет 2 более высокий.

Если работа нашла систему занятой (буфер, и процессор заняты), то работа отбрасывается.

Если пришла работа приоритета 2, буфер пуст, а обрабатывается работа приоритета 1, то пришедшая работа выталкивает менее приоритетную, которая помещается в буфер

Система переходов задает язык

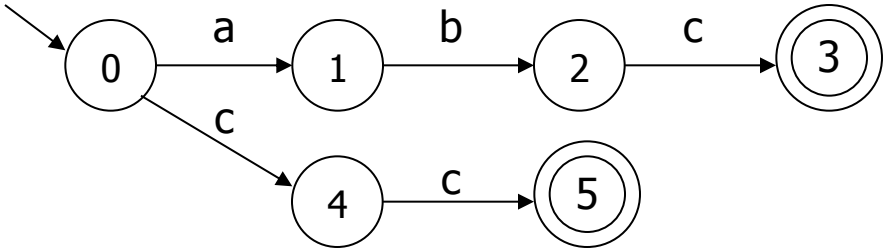


ai – приход работы i-го типа

Построение автоматов для распознавания языков

- Σ_1={a,b,c} L_1={abc, cc}

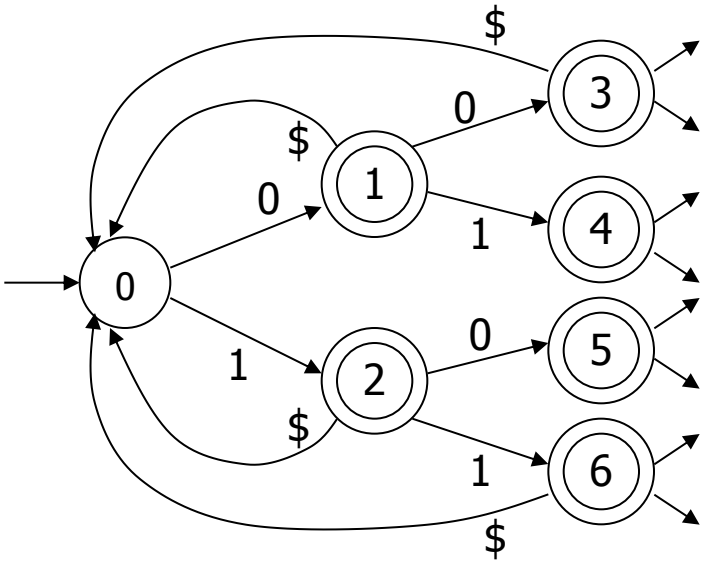
  cc$\in L_1$ cbc$\notin L_1$

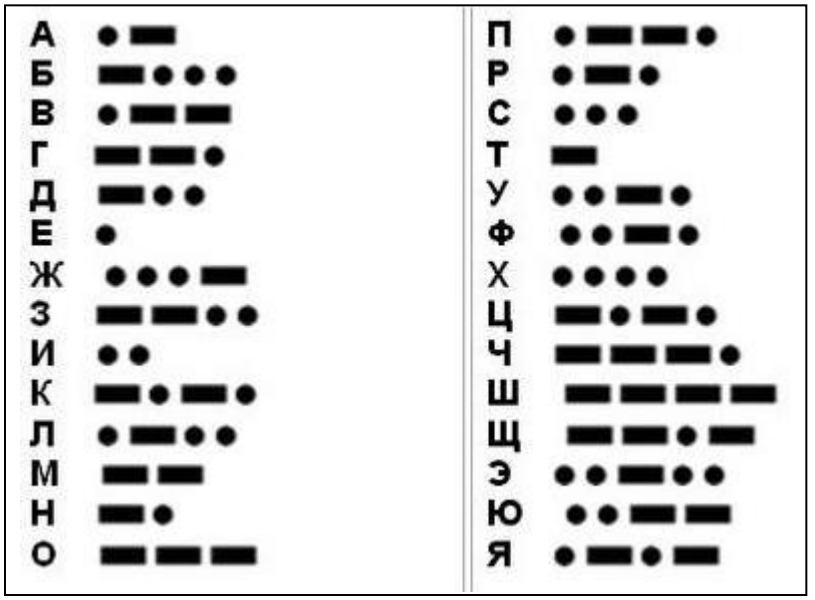


Любой конечный язык может быть распознан конечным автоматом

- Σ={0, 1, $} L = язык азбуки Морзе ($ - разделитель между словами)

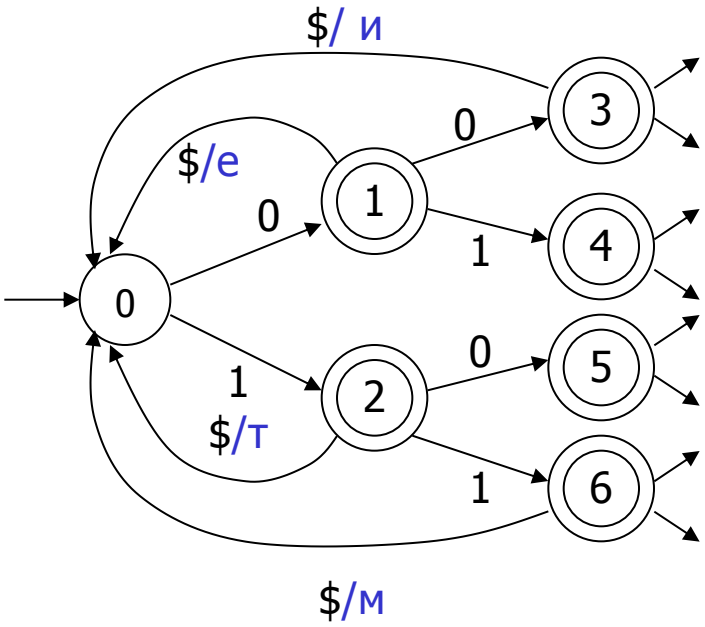
  " . . — . . " – буква э, "— —" – буква м, ...





Автомат с выходом для трансляции языков

- Добавив выходы (семантические действия) к некоторым переходам, можем получить не только распознавание, но и трансляцию



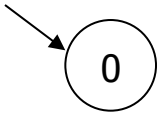
А	●▬	П	●▬▬●
Б	▬●●●	Р	●▬●
В	●▬▬	С	●●●
Г	▬▬●	Т	▬
Д	▬●●	У	●●▬●
Е	●	Ф	●●▬●
Ж	●●●▬	Х	●●●●
З	▬▬●●	Ц	▬●▬●
И	●●	Ч	▬▬▬●
К	▬●▬●	Ш	▬▬▬▬
Л	●▬●●	Щ	▬▬●▬
М	▬▬	Э	●●▬●●
Н	▬●	Ю	●●▬▬
О	▬▬▬	Я	●▬●▬

Мы потом покажем, что трансляция – перевод входной цепочки в некоторый выход – полностью основана на распознавании этой цепочки

Построение автоматов для распознавания языков

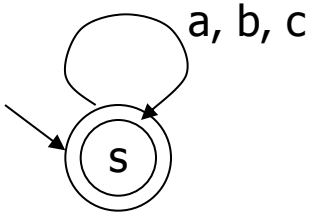
- $\Sigma_2=\{a,b,c\}$ $L_2=\varnothing$

  $cc \notin L1$



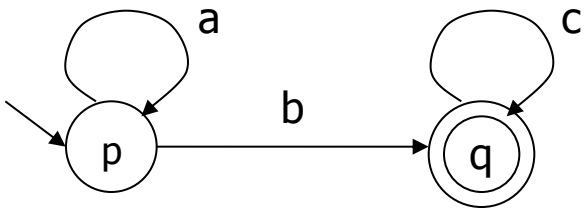
- $\Sigma_3=\{a,b,c\}$ $L_3=\Sigma_3^*$

  $acc \in L_3$, так же, как и любая цепочка из символов a, b, c



- $\Sigma_4=\{a,b,c\}$ $L_5=\{a^n bc^m \mid n, m \geq 0\}$

  $aaabcc \in L_5$ $cbaa \notin L_5$



Если один язык "больше" другого. то это вовсе не значит, что автомат, его распознающий, сложнее. Например, язык L3 - самый "большой", он включает все возможные цепочки, но он распознается очень простым автоматом

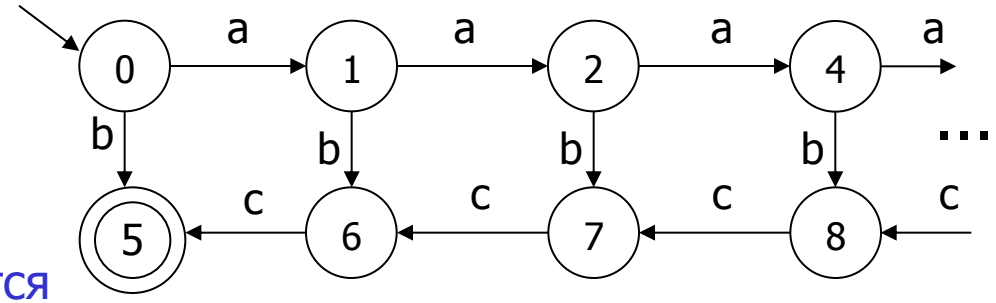
Язык $L_4 \subset L_3$, но распознающий его автомат сложнее

А язык $L_5=\{a^n bc^n\} \subset L_4$, для него вообще нет конечного автомата, его распознающего. Все зависит от структуры цепочек языка!

Лемма о накачке. Доказательство неавтоматности языка

- $\Sigma_5=\{a,b,c\}$ $L_5=\{a^nbc^n \mid n\geq 0\}$

  $aabcc \in L_5$ $cbaa \notin L_5$

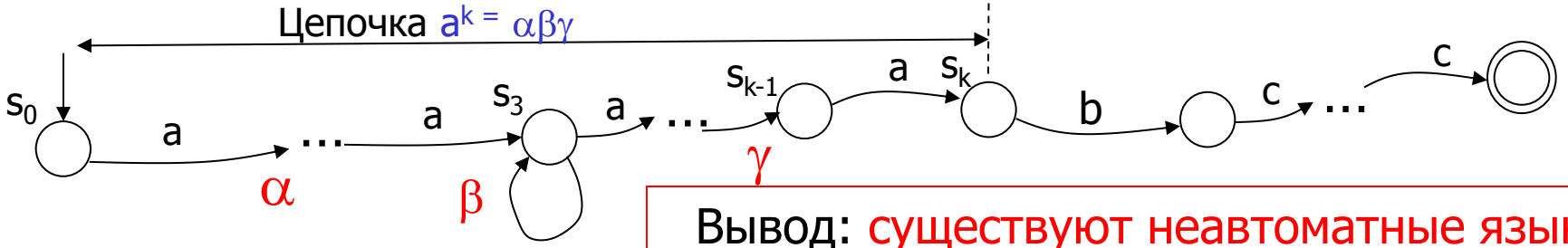


- Определение. Язык L называется автоматным, если существует конечный автомат, допускающий L

- Теорема. Язык $\{a^nbc^n\}$ не автоматный

"Лемма о накачке"

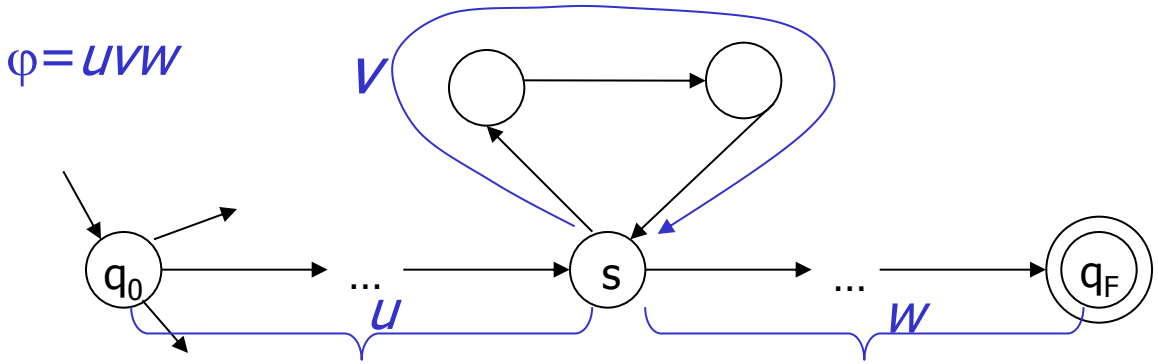
- Доказательство. Предположим противное, т.е. что язык $\{a^nbc^n\}$ автоматный. Тогда существует автомат A, распознающий этот язык. A имеет какое-то число состояний. Пусть A имеет k состояний. Возьмем цепочку $\{a^kbc^k\}$. Поскольку длина цепочки из a = числу состояний, то в автомате A есть состояние, которое цепочка, состоящая только из a, на пути в заключительное состояние проходит дважды, т.е. $\alpha\beta\gamma bc^k \in L_5$. Но если $\alpha\beta\gamma bc^k \in L_5$, то $\alpha\beta^k\gamma bc^k \in L_5$, причем β состоит из a и не пуста. Но уже цепочка $\alpha\beta\beta\gamma bc^k \notin L_5$



Вывод: существуют неавтоматные языки

Лемма о накачке (pumping lemma, лемма о разрастании)

- Пусть хотим доказать, что L, содержащий бесконечное число цепочек, НЕ автоматный
- Предположим противное, т.е. пусть L является автоматным. Тогда существует какой-то конечный автомат A, распознающий L. У него сколько-то состояний. Сколько – мы не знаем. Но знаем точно, что это число конечное. Пусть число его состояний равно k. Выберем в L цепочку φ с подцепочкой v длиной $\geq k$. Эти цепочки мы можем выбрать ПРОИЗВОЛЬНО, так, чтобы доказательство было максимально простым
- Ясно, что на пути, который проходит автомат A при приеме подцепочки φ, какое-то состояние встретится не менее, чем два раза.

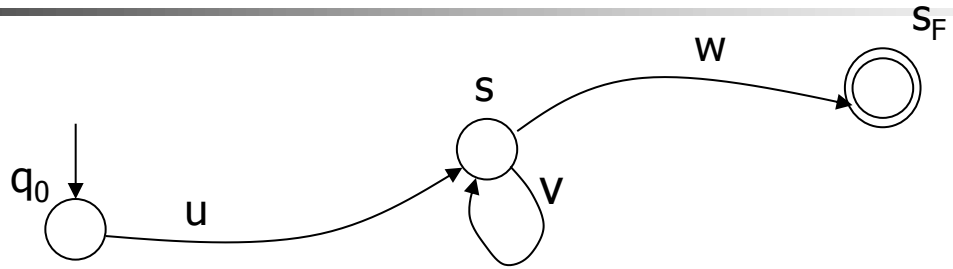


Пример:
число состояний k, и цепочка длиной $\geq k$

Пусть $\varphi = uvw$, где u - цепочка от q_0 до s, v - цепочка из s в s, и w - из s до q_F (финального состояния). Длина цепочки φ: $|\varphi| \geq k$. Цепочка φ обязательно содержит подцепочку, которая переводит A из состояния s в то же состояние s, назовем ее v.

Очевидно, $\varphi = uvw$. Поэтому, наряду с цепочкой uvw, автомат A должен допускать также и uw, uvvw, uvv …vw, и т.д.

Лемма о накачке - формально



- Пусть L – автоматный язык. Тогда существует такое число k (зависящее от L, k-длина "накачки" для L) со следующим свойством: для каждой строки φ языка L длиной *по крайней мере k* строка φ может быть записана как конкатенация трех подстрок u, v и w (φ =uvw), и при этом выполняются следующие условия: 1. $|v|>0$, 2. $|uv|\le k$, 3. для каждого целого $i\ge 0$, $uv^iw\in L$
- Лемма. Если L – автоматный язык, то:

$$(\exists k\in N)(\forall \varphi\in L:|\varphi|\ge k)(\exists u,v,w\in\Sigma^*)[\varphi=uvw \wedge |uv|\le k \wedge |v|\ge 1 \wedge (\forall i\in N\cup\{0\})uv^iw\in L]$$

- Для доказательства неавтоматности L мы используем контрапозицию этой импликации: Если:

$$(\forall k\in N)(\exists \varphi\in L:|\varphi|\ge k)(\forall u,v,w\in\Sigma^* : \varphi=uvw \wedge |uv|\le k \wedge |v|\ge 1)\ [(\exists i\in N\cup\{0\})\ uv^iw\notin L],$$

то язык L неавтоматный.

Проблема: Докажите, что вторая формула – контрапозиция Леммы!

В учебниках почитайте примеры использования Леммы для доказательства неавтоматности языков

Пример доказательства неавтоматности языка

- Термин «накачка» отражает возможность многократного повторения (0, 1, 2, ...) некоторой подстроки в какой-нибудь подходящей строке подходящей длины автоматного языка
- Лемма используется для того, чтобы доказать неавтоматность некоторого языка, имеющего бесконечное число цепочек

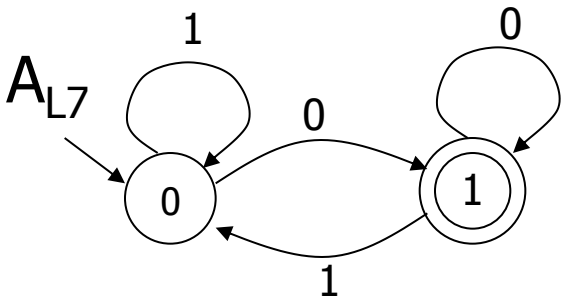
- Пример доказательства. Докажем, что язык $L=\{wcw^R \mid w\in\{a, b\}\}$ – неавтоматный, w^R – обратная цепочка к w (например, для w= abaa w^R=aaba)
- Доказательство от противного. Предположим противное, т.е. что L – автоматный. Тогда существует конечный автомат A, распознающий этот язык. Пусть A имеет k состояний. Возьмем цепочку длиной 2k+1, первая часть ее w имеет длину k. Следовательно, w может быть представлена, как w= $\gamma\mu\nu$, так. что подцепочка μ возвращается в то же состояние, из которого она началась, и $|\mu|\geq 1$.
- В соответствии с Леммой о накачке, если автомат A допускает $wcw^R = \gamma\mu\nu cw^R$, то он допускает и $\gamma\mu\mu\nu cw^R$, и $\gamma\mu\mu\mu\nu cw^R$, и т.д. Но все эти цепочки не принадлежат языку L, поскольку все цепочки в L должны быть симметричными.
- Наше предположение о том, что язык $L=\{wcw^R \mid w\in\{a,b\}\}$ автоматный, приводит к противоречию. Следовательно, этот язык – не автоматный

Построение автоматов для распознавания языков

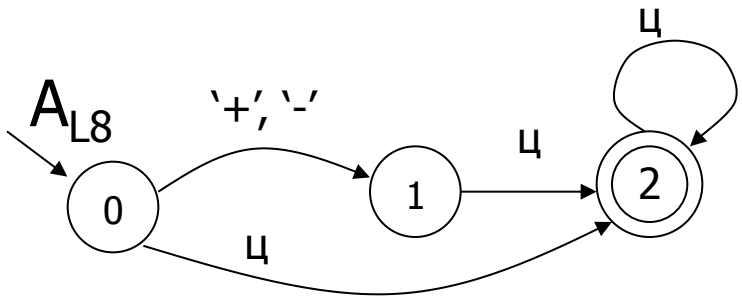
- Σ_6={a,b,c} L_6={$\alpha \in \Sigma_6^*$ | в цепочке α количества вхождений a, b и c равны}
  ccbaba$\in L_6$, ccb$\notin L_6$

Язык L_6 не автоматный

- Σ_7={0,1} L_7=множество четных двоичных чисел 100$\in L_7$ 0111101$\notin L_7$



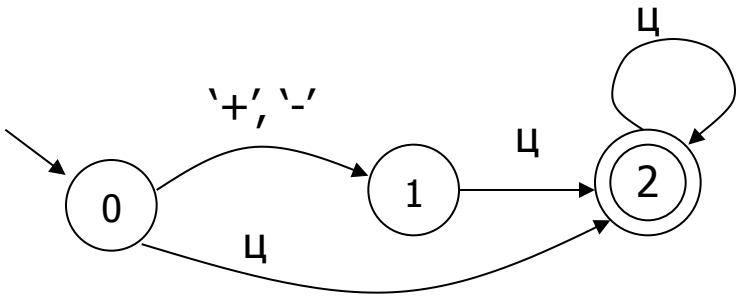
- Σ_8={'+', '-', '0',..., '9'}; L_8= множество целых констант: -67$\in L_8$ 234$\in L_8$ -3$\in L_8$
  +2+3...4$\notin L_8$



- Σ_9={ '(' , ')' } L_9=множество правильных скобочных выражений: (( ))( )$\in L_9$))(( )$\notin L_9$

Язык L_9 не автоматный

Конечные автоматы: переход под воздействием символа из конечного множества символов



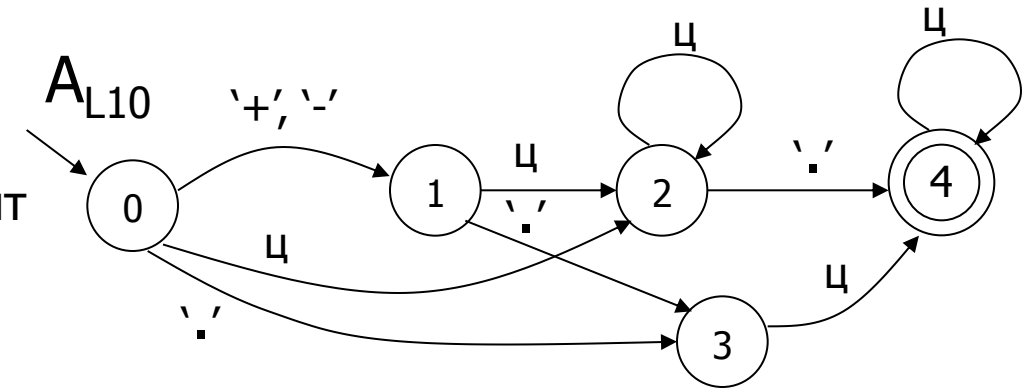
Как понимать переход, помеченный ц?

- Здесь символом ц обозначена любая цифра, фактически, конечное множество символов: ц = {0, 1, 2, …, 9}
- Очевидно, что при реализации КА проверку того, принадлежит ли входной символ конечному множеству символов, выполнить всегда можно
- Например, поскольку кодировки всех цифр идут подряд, то для того, чтобы проверить, принадлежит ли входной символ s множеству ц, достаточно выполнить проверку ‘0’ ≤ s ≤ ‘9’

Построение автоматов для распознавания языков

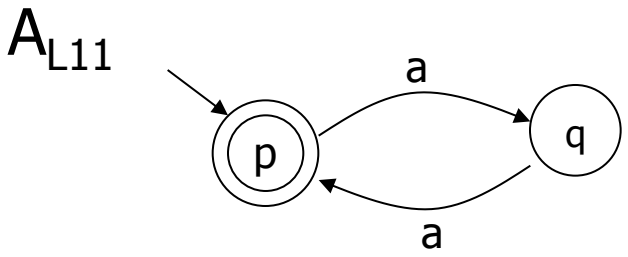
- Σ_{10}={'+', '-', '0',...,'9', '.' } L_{10}= множество вещественных констант

  6.7∈L_{10} 23+.+4.. 6∉L_{10}



- Σ_{11}={a} L_{11}={$\alpha \in \Sigma_{11}^*$ | длина α равна 2k, где k = 0, 1, ...}

  aaaa∈L_{11} a∉L_{11}



- Σ_{12}={a} L_{12}= $\{a^{n^2} \mid n = 0,1,...,\}$

  aaaa∈L_{12} a∈L_{12}, aaa∉L_{12}

Язык L_{12} не автоматный

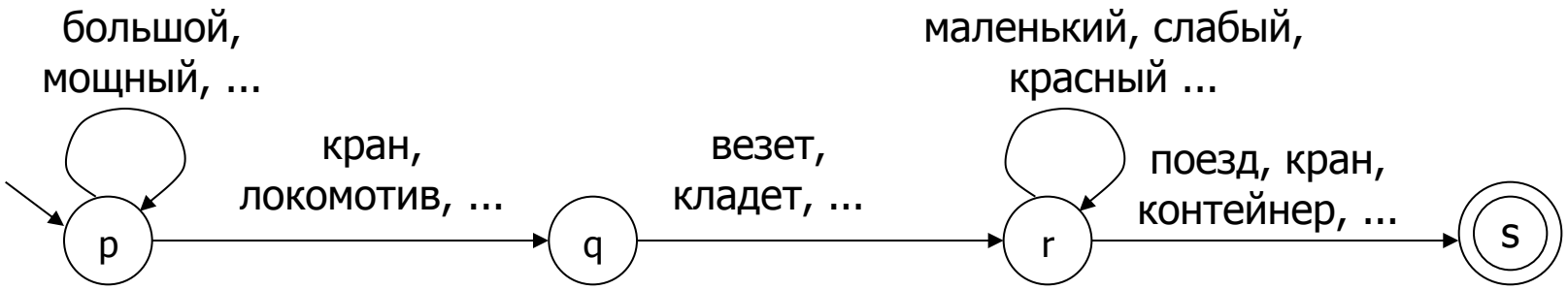
Автоматы для распознавания естественных языков

- Σ_{13}={большой, мощный, тяжелый, …, кран, локомотив, контейнер, …, ставит, ведет, кладет, везет,…} L_{13}= русский язык

Большой тяжелый локомотив везет маленький красный контейнер $\in L_{13}$

Большой кран большой поезд везет тяжелый везет кладет $\notin L_{13}$

- Словарем естественного языка являются словоформы – слова с их изменениями по родам, числам, падежам и проч. Всех словоформ – конечное число, а предложений языка – бесконечное число. Примеры словоформ: большой, ставит, кран, кладет везет, везли, везут, … . Это все – различные символы конечного словаря для русского языка. Т.о. цепочками естественного языка можно считать его предложения
- Русский язык (и другие естественные языки) НЕ являются автоматными
- Небольшие фрагменты естественных языков можно задать конечными автоматами



Языки программирования высокого уровня

- Σ_{14}={‘+’, ‘-’, … , ‘0’,…, ‘9’, …, begin, end, if, …, }; L_{14}= язык Паскаль

  end begin if :: begin 0:== ; $\notin L_{14}$ begin int a; end $\in L_{14}$

  - Языки программирования высокого уровня не являются автоматными: они содержат вложенности конструкций произвольной глубины
  - Небольшие фрагменты языков программирования являются автоматными
  - Это:
    комментарии
    идентификаторы (имена)
    служебные слова (begin, while, for, end, …)
    константы
  - Обычно обработка этих фрагментов выполняется на первых проходах трансляторов, где выбрасываются комментарии, пробелы, выделяются лексемы
  - Лексемы -минимальные единицы языка, имеющие смысл
  - Первые проходы трансляторов строятся на основе теории конечных автоматов - распознавателей

Язык Милан

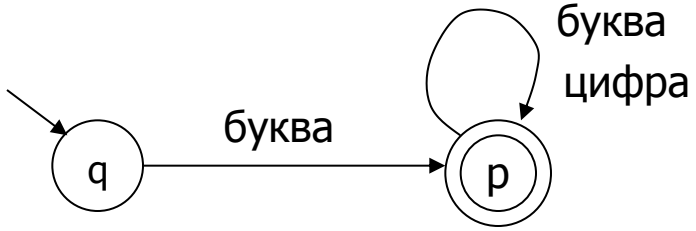
Σ_{15}={'+', '-', ... , '0',..., '9', ..., begin, end, if, ..., }; L_{15}= язык MiLan (Mini Language)

"end begin if :: begin 0:== ;" $\notin L_{15}$

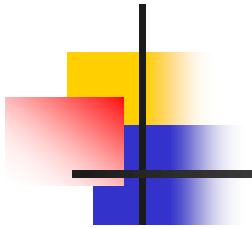
"begin
a :=5;
input (x);
a :=x+a*3;
while x>0 do a:=a*2; x:=x-1
end" $\in L_{15}$

- Язык Milan не автоматный: в частности, он содержат вложенности операторов произвольной глубины

- Идентификатор – это любая последовательность букв и цифр, начинающаяся с буквы



Это – автоматный подъязык языка Милан



- Дополнительный материал:
Теорема Майхилла-Нероуда об автоматных языках

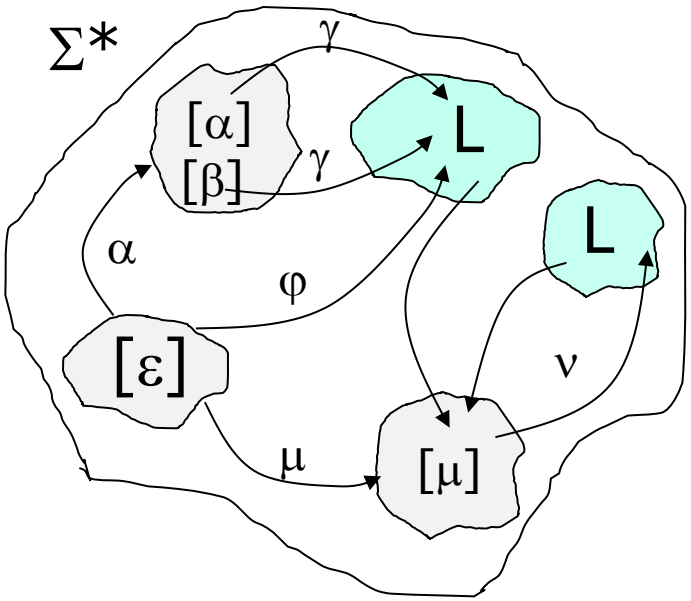
Когда множество цепочек будет автоматным языком? Отношение эквивалентности на множестве цепочек

Правым инвариантом называется такое отношение эквивалентности R на Σ^*, что:

$$(\forall \alpha,\beta \in \Sigma^*)\ \alpha R \beta \Rightarrow (\forall \gamma \in \Sigma^*)\ \alpha\gamma R \beta\gamma$$

Пусть L – язык над словарем Σ. Правый инвариант R_L, ассоциированный с L:

$$(\forall \alpha,\beta \in \Sigma^*)\ \alpha R_L \beta \Leftrightarrow (\forall \gamma \in \Sigma^*)\ (\alpha\gamma \in L \Leftrightarrow \beta\gamma \in L)$$



Классы эквивалентности R_L

$$\delta([\alpha], a) = [\alpha a]$$

- Классы эквивалентности R_L – это множество таких предысторий (цепочек), что α и β попадают в один класс ($[\alpha] = [\beta]$), если для любой цепочки γ либо обе цепочки $\alpha\gamma$ и $\beta\gamma$ принадлежат L, либо обе не принадлежат L
- На рисунке: $[\varepsilon]$ – класс эквивалентности, содержащий пустую цепочку;
- L – класс эквивалентности, содержащий цепочки языка L;
- φ - цепочка принадлежит языку L;
- $[\mu]$ – класс эквивалентности цепочек, не принадлежащих L (любое продолжение не приведет в L). Соответствующее состояние A_L некодостижимо

Теорема Майхилла-Нероуда

- ***Теорема Майхилла-Нероуда***. (Джон Майхилл, Энил Нероуд, 1958 г.)
  Пусть L – язык над словарем Σ. Следующие три утверждения эквивалентны:

  1. Язык L– автоматный, т.е. существует КА допускающий L.

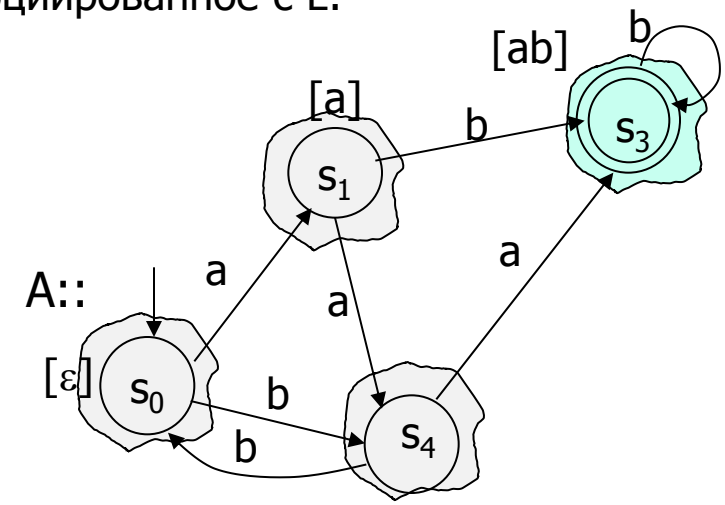
  2. Существует правый инвариант R с конечным индексом (конечным числом классов эквивалентности) на Σ^*, такой, что $L = \cup_{[x]\in\Pi}[x]_R$, где П – некоторое множество классов эквивалентности R

  3. Существует отношение R_L конечного индекса, ассоциированное с L:
  $(\forall\alpha,\beta\in\Sigma^*)\ \alpha R_L\beta \Leftrightarrow (\forall\gamma\in\Sigma^*)\ (\alpha\gamma \in L \Leftrightarrow \beta\gamma \in L)$

Доказательство (1 и 3).

Пусть R_L имеет классы эквивалентности $[\alpha_1]$, …, $[\alpha_n]$.
Определим конечный автомат A_L так:
- множество состояний $S = \{ [\alpha_1], [\alpha_2], \ldots, [\alpha_n] \}$.
- начальное состояние: $[\varepsilon]$ (класс, включающий ε),
- функция переходов $\delta([\alpha], a) = [\alpha a]$,
- множество финальных состояний $F=\{[\alpha] \mid \alpha\in L\}$

Очевидно, что $L_A=\{ \alpha \mid \delta([\varepsilon], \alpha)\in F \}$

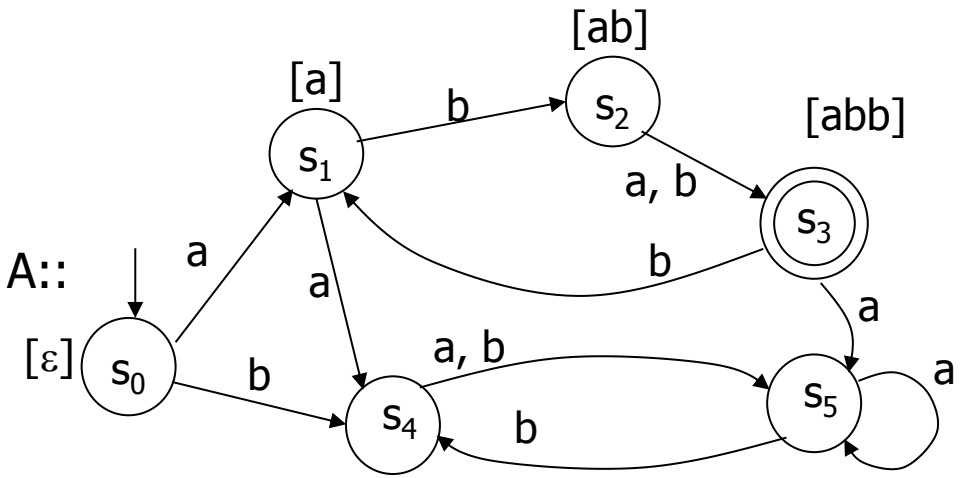


Состояния – классы эквивалентных предысторий.
Эквивалентность – относительно дальнейшего поведения: допущения цепочек языка L

Теорема Майхилла-Нероуда (2)

Пример отношения R – правого инварианта с конечным числом классов эквивалентности (конечного индекса) и соответствующего ему конечного автомата.

Классы эквивалентности R:
({ε}, {b,aa,bbb, ababa, …}, {a, abbb, … }, … - соответствуют состояниям



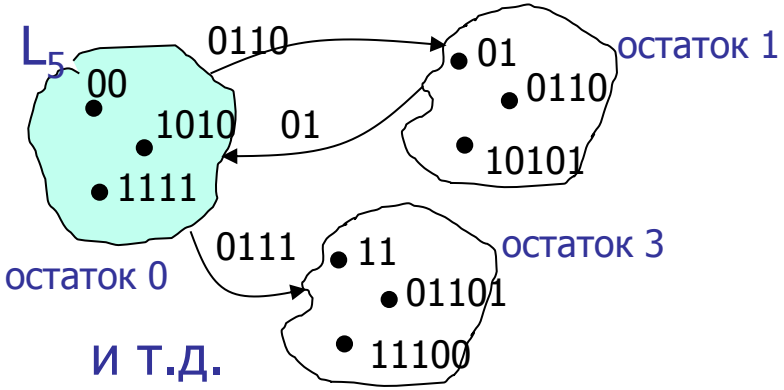
aRabbb, bRaa, abRabbbb, …

s_0 :	{ε}	= [ε]
s_1 :	{a, abbb, …}	= [a]
s_2 :	{ab, abbbb, …}	= [ab]
s_3 :	{aba, abab, …}	= [aba]
s_4 :	{b,aa,bab,aabb, …}	= [aa]
s_5 :	{bb, aab, bba, …}	= [bb]

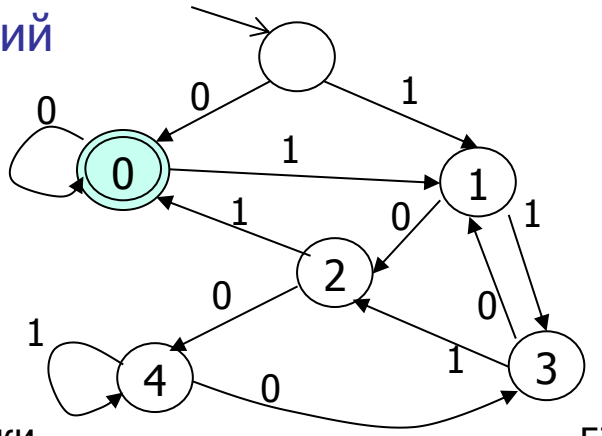
Теорема Майхилла Нероуда определяет необходимое и достаточное условия автоматности языка.
Она также позволяет доказать, что данный язык не автоматный.

Пример эквивалентности, ассоциированной с языком над Σ, и соответствующего конечного автомата

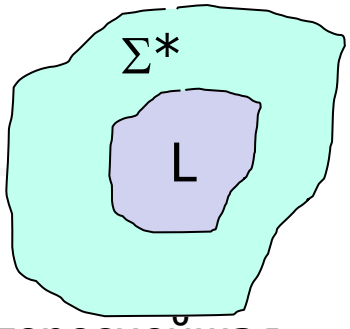
- Одно из простейших отношений эквивалентности – эквивалентность по модулю, (например, по модулю 5) на множестве неотрицательных целых чисел. Назовем его π_5
- Числа A и B находятся в отношении π_5 , если остатки от их деления на 5 равны. Отношение π_5 имеет 5 классов эквивалентности: {0, 5, 10, 15, 20, … } – остаток от деления на 5 = 0, {1, 6, 11, 16, 21, … } – остаток от деления на 5 = 1 и т.д.
- Пусть $[a]_\pi$ обозначает такой класс эквивалентности π_5, которому принадлежит a. Поэтому в десятичной системе счисления $[0]_{\pi 5}=[10]_{\pi 5}=[325]_{\pi 5}$, $[3]_{\pi 5}=[333]_{\pi 5}=[103]_{\pi 5}$
- Пусть Σ={0, 1} и пусть отношение ρ_5 на Σ* объединяет цепочки, представляющие те числа в двоичной системе счисления, которые при делении на 5 дают один и тот же остаток. Ясно, что ρ_5 – эквивалентность на Σ* с конечным индексом. Пусть язык L_5 – это множество непустых цепочек над Σ, которые представляют двоичные числа, делящиеся без остатка на 5. Ясно, что ρ_5 имеет 6 классов эквивалентности. Одному из них принадлежит только пустая цепочка ε, остальные объединяют числа с одним и тем же отношением делимости по модулю 5



Соответствующий КА

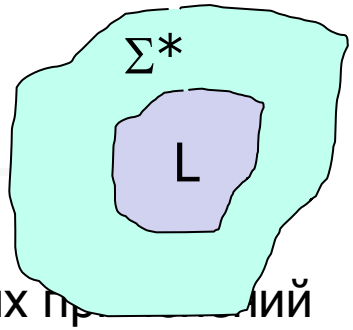


Заключение

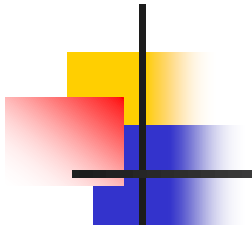


- Зачем изучать теорию формальных языков (мотивация)?
  - Теория формальных языков – это “цветок информатики”, интереснейшая область математики, имеющая приложения в построении трансляторов
  - Каждый специалист в области информатики будет сталкиваться с необходимостью разработки языков интерфейсов, языков представления данных, формул, ... Новые языки разрабатываются постоянно во всех инженерных областях при их автоматизации, для них нужно строить свои трансляторы
- Что такое формальный язык? – Это просто множество цепочек
  - Формальный язык – это множество (обычно бесконечное) конечных цепочек (слов, предложений) над конечным словарем
  - Для задания бесконечных объектов математика строит конечные модели
- Что такое грамматика? – Это конечный формализм для задания языка
  - Существует два способа задания языка: распознавание и порождение. Трансляция входной цепочки основана на распознавании этой цепочки, т.е. на алгоритме проверки того, что эта цепочка принадлежит языку

Заключение (2)



- Конечные автоматы как формализм задания языков
  - КА имеют множество применений. Одно из самых интересных применений – в области формальных языков. КА используются здесь как распознаватели “правильных” входных цепочек над конечным словарем. Такие автоматы не имеют выхода: выходом в них является тип состояния, в котором находится автомат (допускающее либо недопускающее)
  - Входная цепочка распознается автоматом, если она переводит его из начального в одно из финальных (допускающих) состояний
  - Каждый конечный автомат-распознаватель задает какой-нибудь язык. Язык, для которого существует распознающий его автомат, называется автоматным. Конечных автоматов распознавателей бесконечное число, автоматных языков тоже бесконечное число
- Не все языки являются автоматными
  - Не все формальные языки можно задать с помощью КА. Автоматные языки – очень слабый подкласс формальных языков. Язык Милан, для которого мы будем строить транслятор, не автоматный. Неавтоматным является любой язык с вложенными конструкциями
  - Лемма о накачке - удобный метод определения того, что заданный язык не является автоматным



Спасибо за внимание