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

Структура курса

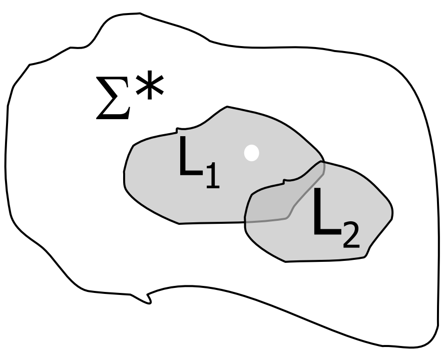
- Конечные автоматы-распознаватели – 4 л
  - Лекция 1. Формальные языки. Примеры языков. Грамматики. КА
  - Лекция 2. Теория конечных автоматов-распознавателей
  - Лекция 3. Трансляция автоматных языков
  - Лекция 4. Регулярные множества и регулярные выражения
- Порождающие грамматики Хомского – 3 л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

Регулярные множества

- Регулярные множества – это (бесконечные) множества цепочек, построенных из символов конечного словаря по определенным правилам

- Вспомним определение языка:

Σ={a,b,c} – словарь; Σ^*={ε, a, b, c, aa, ac, ab, ba, cbba, ... }; $L \subseteq \Sigma^*$

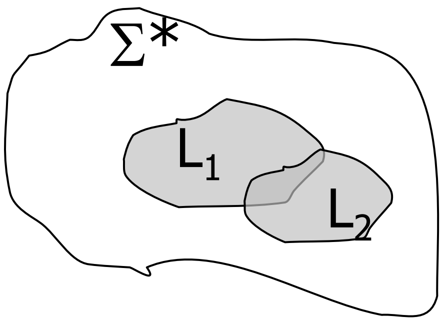


- Словарь конечен и все цепочки конечны, но множество Σ^* бесконечно
- Определение. Языком над конечным словарем Σ называется любое подмножество Σ^*, $L \subseteq \Sigma^*$
- Обычно число цепочек языка бесконечно
- Число возможных языков над конечным словарем несчетно – потому что несчетно число подмножеств счетного множества

Регулярные множества, как (бесконечные) множества конечных цепочек – это языки. Их называют также регулярными языками

Как строятся регулярные языки? Три операции над множествами цепочек

- Σ={a, b, c} – словарь; Σ^*={ε, a, b, c, aa, ac, ab, ba, cbba, ... }



Конкатенация (произведение) $L_1 \cdot L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\}$
Пример: {abbc, cb}·{ba, ccc} = {abbcba, abbccc, cbba, cbccc}
Обозначим $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = L \cdot L$, $L^{k+1} = L \cdot L^k$

Объединение $L_1 \cup L_2 = \{\alpha \mid \alpha \in L_1$ или $\alpha \in L_2\}$
Пример: {abbc, cb} ∪{ba, ccc} = {abbc, cb, ba, ccc}

Итерация $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \cup^{\infty}_{k=0} L^k$ (0 или больше раз)
Пример: {abb, c}*={ε, abb, c, abbabb, abbc, cc, cabb, abbccabbabb, abbcc, ... }

Итерация языка L - множество цепочек L*, каждая из которых представляет собой 0 или большее число конкатенаций любых цепочек из L (звезда Клини)

Усеченная итерация $L^+ = L^1 \cup \ldots = \cup^{\infty}_{k=1} L^k$ (1 или больше раз) $L^* = \{\varepsilon\} \cup L^+$
Пример: {abb, c}⁺={abb, c, abbabb, abbc, cc, cabb, abbabbccabb, abbcc, ... }

Регулярные множества. Формальное определение

- **Определение.** Класс регулярных множеств над конечным словарем Σ определяется так:

  $\varnothing$ - регулярное множество;

  $\{\varepsilon\}$ - регулярное множество;

  $\{a\}$ - регулярное множество для любого $a \in \Sigma$;

- Если P и Q - регулярные множества, то регулярны и

  объединение $P \cup Q$;

  произведение (конкатенация) $P \bullet Q$ (часто записывается, как PQ);

  итерация P^* ;

- Примеры регулярных множеств:

  $\{ab, ba\}^* \bullet \{aa\} = \{aa, abaa, baaa, ababbaaa, babaabaa, \ldots\}$

  $\{b\} \bullet (\{c\} \cup \{d, ab\}^*) = \{bc, b, bd, bab, bababd, \ldots\}$

- Примеры нерегулярных множеств:

  $\{a^n b^n \mid n>0\} = \{ab, aabb, aaabbb, aaaabbbb, \ldots\}$;

  $\{\alpha \mid$ в цепочке α количества вхождений символов a и b совпадают$\}$ = $\{aababbba, \varepsilon, baababaabb, ba, abba, \ldots\}$

Регулярные выражения – формулы для представления регулярных множеств

- Регулярные множества хороши тем, что их можно описать формулами, которые называются регулярными выражениями.
- Регулярное выражение показывает, как можно построить данное регулярное множество цепочек с использованием трех операций: конкатенации, объединения и итерации из одноэлементных множеств
- **Определение.** Класс регулярных выражений над конечным словарем Σ определяется так:

$\varnothing$, ε - и любое $a \in \Sigma$ - регулярные выражения;

- Если R1 и R2- рег. выр., то регулярными выражениями будут
  - их сумма R1 + R2; (знак '+' часто записывают как '|')
  - их произведение R1R2;
  - итерация R1* и урезанная итерация $R1^+$;
- Приоритеты: *, произведение, сложение **(* - УНАРНАЯ операция!!!)**
- Скобки ( ) используются для группировки (для задания порядка выполнения операций)

Примеры регулярных выражений: ab + ba*; (aa)*b + (c + dab)*

Регулярные множества и регулярные выражения – различные сущности

- Регулярные множества и регулярные выражения весьма близки. Но они являются разными сущностями
- Регулярное множество - *это множество цепочек* (в общем случае бесконечное), и, как любое множество цепочек, его можно считать языком (регулярным языком)

Например, $\{b\}\bullet(\ \{c\} \cup (\ \{d\} \cup \{a\}\ )^*\ ) = \{bc, b, bd, ba, \ldots, baddadddda, \ldots\}$

- Регулярное выражение - *это формула, схематично показывающая, как можно построить соответствующее ей регулярное множество с помощью перечисленных трех операций* (и эта формула всегда конечна), например,

$$b\ (\ c + (\ d + a\ )^*)$$

$M=\{bc, b, bd, ba, bdd, baa, bdadd, \ldots\}$	Регулярное множество M: множество цепочек, полученное по определенным правилам
$M = \{b\}\bullet(\ \{c\} \cup (\ \{d\} \cup \{a\}\ )^*\ )$	Последовательность применения операций к одноэлементным множествам для получения M
$R_M = b\ (\ c + (d + a)^*\ )$	Регулярное выражение R_M: формула, показывающая порядок применения операций для получения M

Регулярные множества и регулярные выражения

- Регулярное выражение *- это формула, конечная схема, шаблон, по которому строится (в общем случае, бесконечное) множество цепочек*
- Соответствие операций регулярных выражений и регулярных множеств:
  - конкатенация подформул соответствует произведению множеств
  - сложение '+' (или) соответствует операции $\cup$ объединения множеств
  - итерация * - унарная операция, соответствует операции M_R* итерации соответствующего множества (т.е. 0 или большее число повторений)
- Формула b(c + (d + ab)*) определяет регулярный язык {bc, b, bd, bdd,... }, полученный последовательностью операций над одноэлементными множествами:

$$\{b\}\bullet(\ \{c\} \cup (\{d\} \cup \{a\}\bullet\{b\})^*)$$

- Регулярные множества (регулярные языки) определяют семантику, значение регулярных выражений. Два регулярных выражения равны, если определяющие их регулярные множества

Итак, некоторый класс бесконечных множеств цепочек может быть задан конечными формулами – регулярными выражениями. Это те множества, которые можно построить конечным применением 3-х операций (конкатенация, объединение, итерация), начиная от одноэлементных множеств

Регулярные выражения и регулярные множества

$b\ (c + (d + a\,b)^*)$

регулярное выражение (формула F)

$\{b\} \bullet (\{c\} \cup (\{d\} \cup \{a\} \bullet \{b\})^*)$

последовательность операций над одноэлементными множествами символов, дающая некоторый язык L

{bc, b, bd, ba, bdd, baa, bdadd, ...}

множество цепочек (язык), полученное по правилам, которые задаются формулой $b(c + (d + ab)^*)$

Примеры: регулярные множества как семантика регулярных выражений

Рег. выражение	Соответствующий регулярный язык – “значение” рег выр
a + b + cbbab	Три цепочки: a, b и cbbab, т.е. язык {a, b, cbba}
ba*	Все цепочки, начинающиеся с b, за которым идет какое-то число символов a (в том числе и 0)
a* b a* b a*	Все цепочки из a и b, которые содержат точно два вхождения b
$(a + bb)^+$	Все непустые цепочки из a и b, в которых b входят только парами
(a+b)*(aa+bb) (a+b)*	Все цепочки из a и b, которые содержат по крайней мере одну пару рядом стоящих a или b
(0 + 1)* 10010	Все цепочки из 0 и 1, которые оканчиваются цепочкой 10010
$(0+1+2+3+4+5+6+7+8+9)^+$	Все целые константы без знака (все конечные цепочки, которые содержат 1 или большее число цифр)

Иногда (не совсем правильно) регулярное выражение отождествляют с соответствующим регулярным языком: *Задан регулярный язык 10*1* *вместо* *Задан язык, определяемый регулярным выражением 10*1*

Эквивалентность регулярных выражений

- Два регулярных выражения R и S называются эквивалентными (обозначается $R=S$) тогда и только тогда, когда соответствующие им языки совпадают, т.е. $L(R)=L(S)$
- Пример:
  $a^*a^+ = aa^* = a^+$ - все непустые цепочки, состоящие из a

Для любых регулярных выражений R, S и T справедливы соотношения:

$R + S = S + R$; $R + R=R$; $\varnothing + R = R$; $(R + S) + T = R + (S + T)$

$\varepsilon R=R\varepsilon = R$; $(RS)T=R(ST)$; $\varnothing R=R\varnothing=\varnothing$; но в общем случае $RS\neq SR$

$R(S+T)=RS+RT$; $(S+T)R=SR+TR$

$R^*=\varepsilon + R + R^2 + R^3 + .. + R^kR^*$;

$RR^*=R^*R$; $R(SR)^*=(RS)^*R$

$R^+=R + R^2 + R^3 + ... + R^kR^*$;

$R^+ + \varepsilon = R^*$

Пример эквивалентных преобразований

- $b(b + aa^*b) \quad =$
  - по правилу $R = \varepsilon R$

  $b(\varepsilon b + aa^*b) \; =$
  - по правилу $SR + TR = (S + T) R$

  $b(\varepsilon \; + aa^*)b =$
  - по правилу $RR^* = R^+$

  $b(\varepsilon \; + a^+)b \quad =$
  - по правилу $\varepsilon + R^+ = R^*$

  $b(a^*)b \qquad\qquad =$
  - убираем скобки

  ba^*b

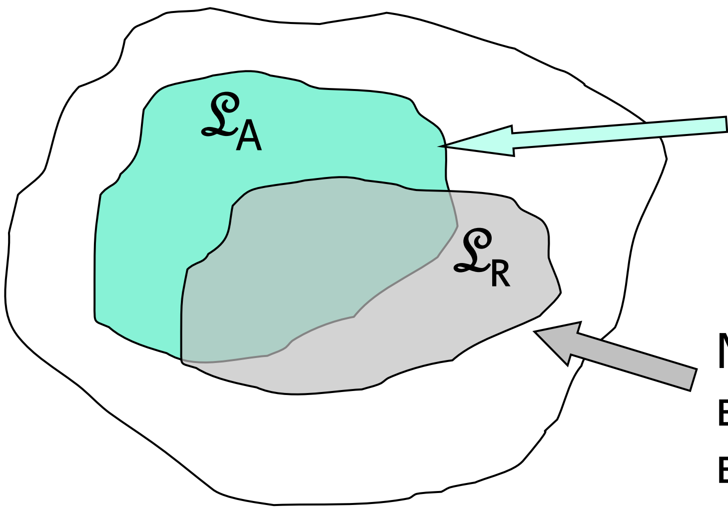
- Отсюда

  $$b(b + aa^*b) \; = \; ba^*b$$

Канонического представления регулярного выражения в классе выражений не существует

Регулярные языки и автоматные языки

- Итак, регулярные выражения - это конечные формулы, задающие множества цепочек - регулярные языки. Но подобным же свойством обладают и конечные автоматы - они тоже задают языки



Множество автоматных языков с алфавитом Σ (определяются конечными автоматами с множеством входных символов Σ)

Множество регулярных языков (определяется всеми возможными формулами – регулярными выражениями над алфавитом Σ)

- Как соотносятся между собой классы языков, задаваемые конечными автоматами и регулярными выражениями?

Например, можно ли построить КА, распознающий регулярное множество цепочек {bc, b, bd, bdd, bda, baad, ... }, которое задается регулярным выражением b(c + (d + a)*)

Теорема Клини

- Теорема Клини. Классы регулярных множеств и автоматных языков совпадают, т.е. $\mathfrak{L}_A = \mathfrak{L}_R$. Автоматный язык часто называют регулярным

Это значит, что 1) любой язык, распознаваемый КА, может быть задан формулой, и 2) для любого языка, заданного формулой (регулярным выражением), может быть построен распознающий его КА

- Доказательство - в два этапа:

ЭТАП 1 Докажем, что любой автоматный язык является регулярным (его можно задать регулярным выражением): $L \in \mathfrak{L}_A \Rightarrow L \in \mathfrak{L}_R$

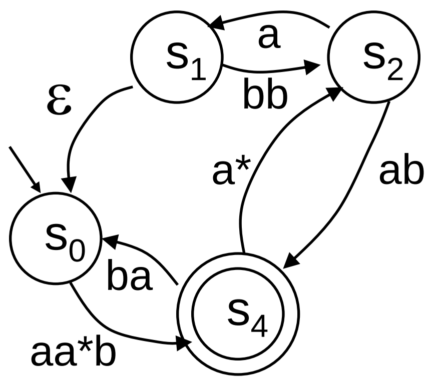
Покажем, что по конечному автомату A всегда можно построить регулярное выражение, формулу, описывающую тот язык, который допускает A

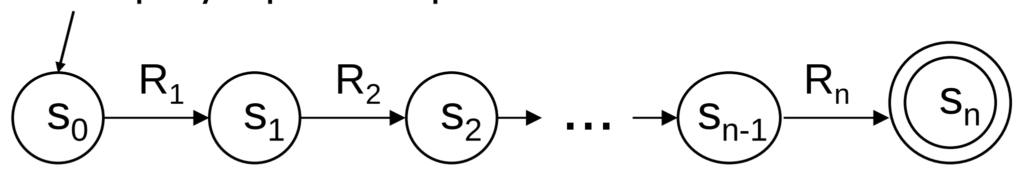
ЭТАП 2 Докажем, что любой регулярный язык является автоматным (его можно задать конечным автоматом): $L \in \mathfrak{L}_R \Rightarrow L \in \mathfrak{L}_A$

Покажем, что по формуле – регулярному выражению R, всегда можно построить конечный автомат, который допускает тот язык, который задает R

ЭТАП 1: Конечный автомат $\Rightarrow$ Регулярное выражение

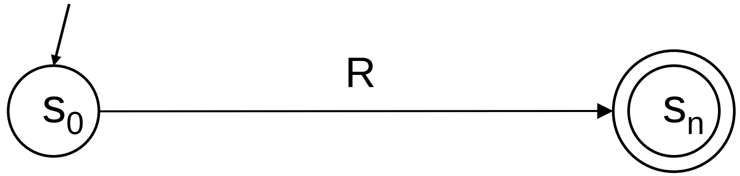
- По автомату A будем строить обобщенную систему переходов T_A, допускающую тот же язык. На переходах T_A будут стоять регулярные выражения





- Обобщенная система переходов T допускает все те цепочки, которые принадлежат языку $L(R_1R_2...R_n)$

- **Если** наш автомат представлен системой переходов с двумя состояниями:



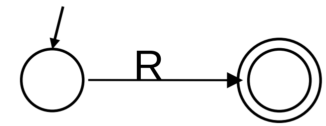
то язык, который допускает этот автомат, задается регулярным выражением R

- Наша задача – привести любой конечный автомат к этому виду

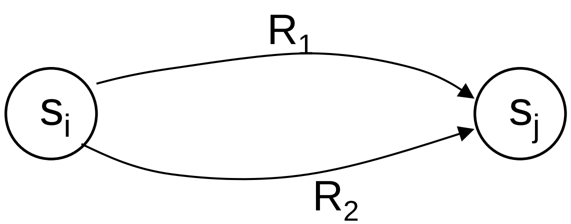
Конечный автомат ⇒ Регулярное выражение (2)

Представим КА в нормальном виде системы переходов, т.е.

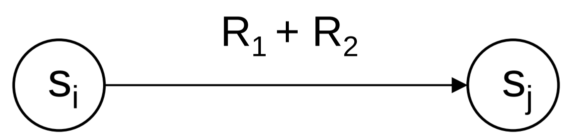
Рассмотрим правила преобразования от этой формы к такой:



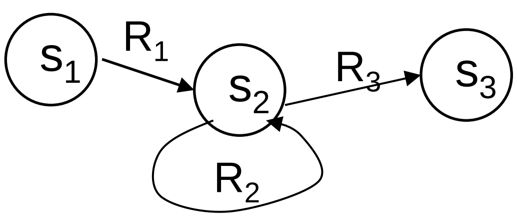
- К системе переходов A будем применять в любом порядке два правила редукции, которые не изменяют допускаемый системой переходов язык
  - Редукция дуг



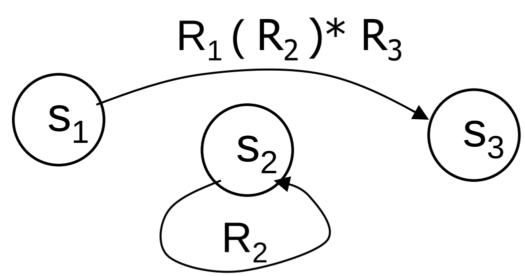
⇒



  - Редукция вершин

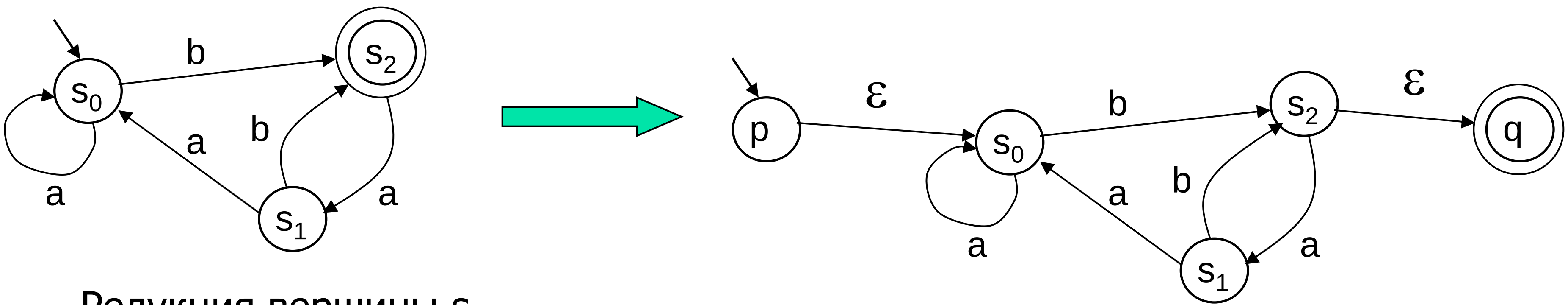


⇒

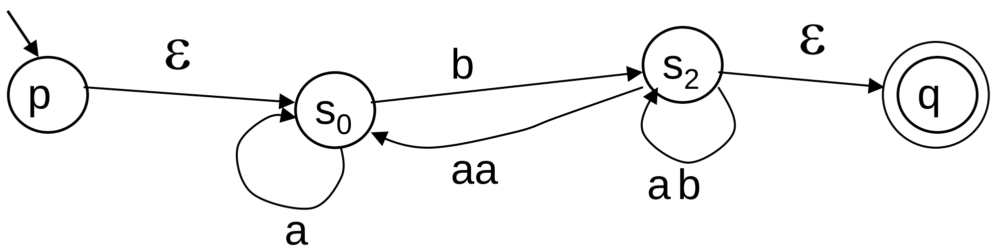


- Применение в любом порядке этих правил редукции приведет к желаемой структуре

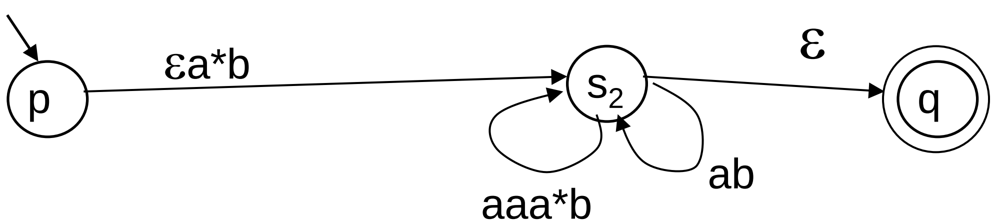
Пример: Конечный автомат
⇒ Регулярное выражение



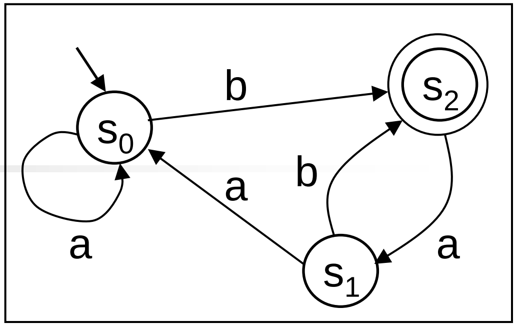
- Редукция вершины s_1

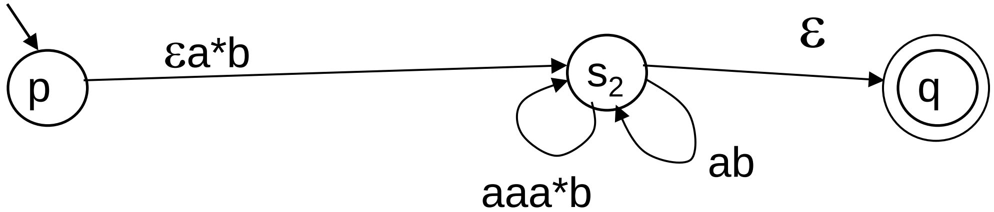


- Редукция вершины s_0

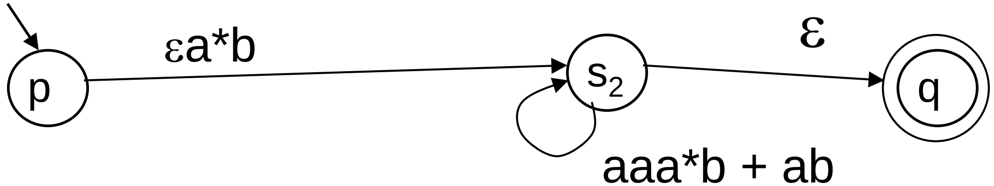


Пример: Конечный автомат ⇒ Регулярное выражение (2)

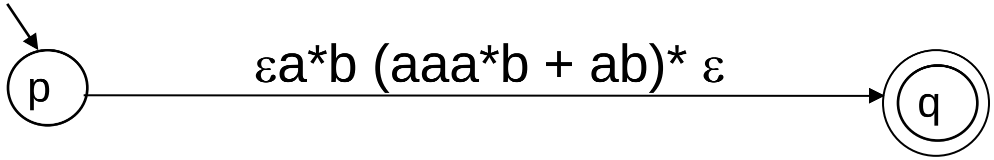




- Редукция дуги



- Редукция вершины s_2



- $R_A = \varepsilon a^*b\ (aaa^*b + ab)^*\ \varepsilon$

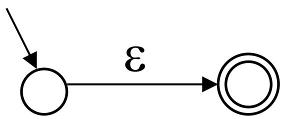
ЭТАП 2: Регулярное выражение ⇒ Конечный автомат

Пусть задано регулярное выражение R

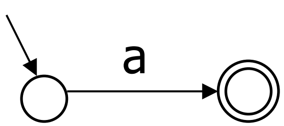
По структуре R строим конечный автомат:

- R= ∅ A_R =

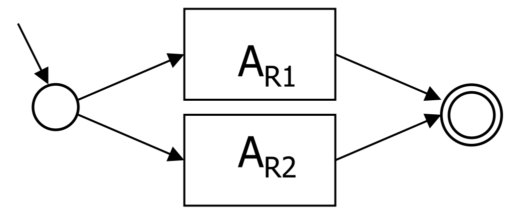
- R= ε A_R =



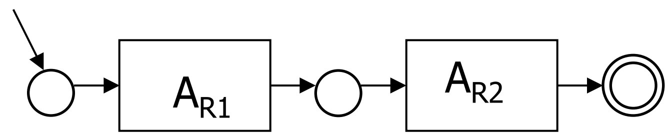
- R= a A_R =



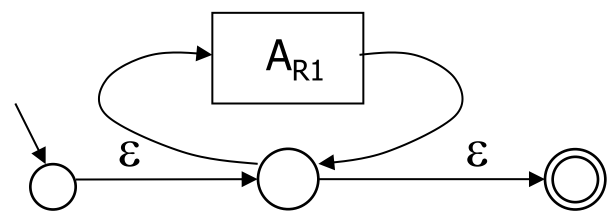
- R= R1 + R2 A_R =



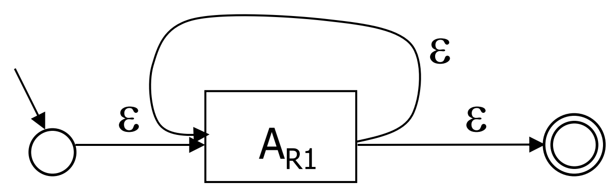
- R= R1 R2 A_R =



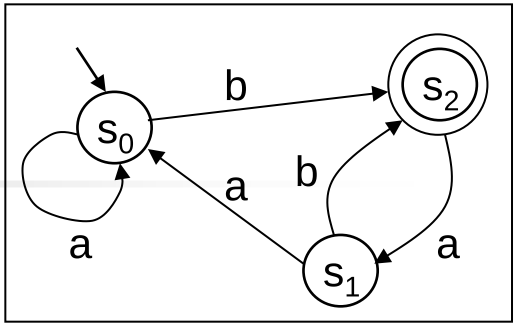
- R= R1* A_R =



- R= R1$^+$ A_R =

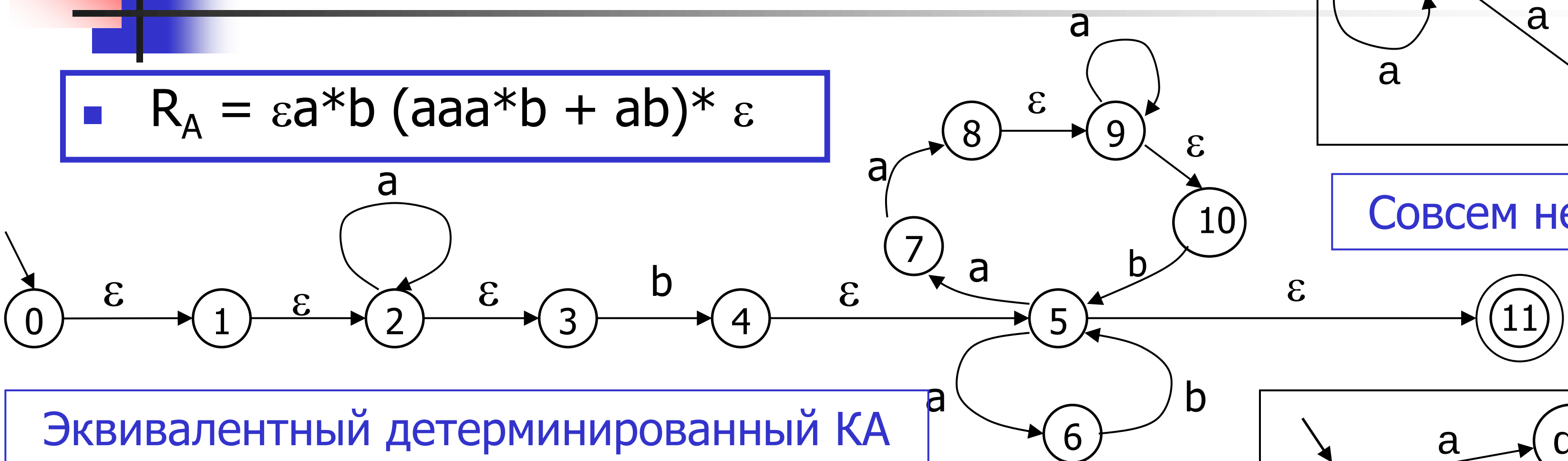


Пример: Регулярное выражение ⇒ Конечный автомат



Совсем не похож

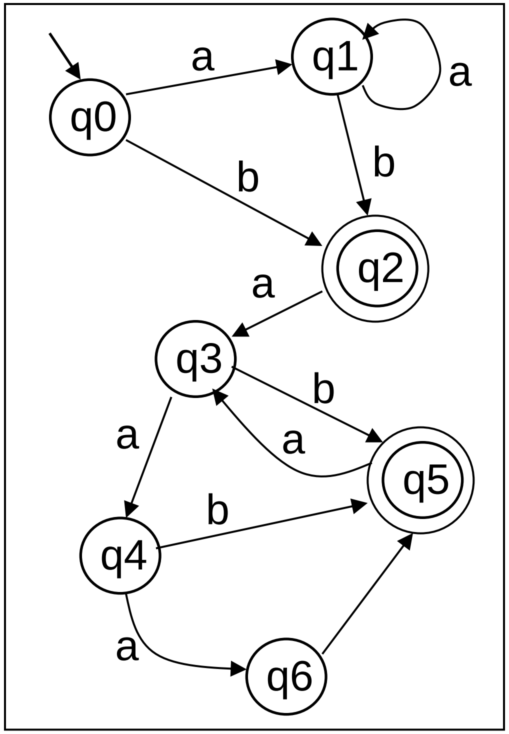
- $R_A = \varepsilon a^*b\,(aaa^*b + ab)^*\,\varepsilon$



Эквивалентный детерминированный КА

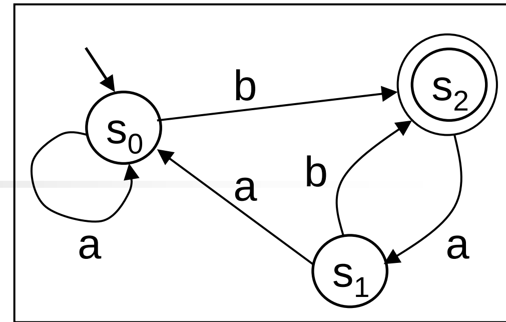
	δ	x=a	x=b
$q0^-$	$\{0^-, 1, 2, 3\}^-$	{ 2, 3 }	{ 4, 5,11 }
q1	{ 2, 3 }	{ 2, 3 }	{4, 5, 11}
$q2^+$	$\{4, 5, 11\}^+$	{ 6, 7 }	
q3	{ 6, 7 }	{ 8,9,10}	{ 5, 11 }
q4	{ 8, 9, 10 }	{ 9, 10 }	{ 5, 11 }
$q5^+$	$\{5, 11\}^+$	{ 6, 7 }	
q6	{ 9, 10 }	{ 9, 10 }	{ 5, 11 }

δ	x=a	x=b
$q0^-$	q1	q2
q1	q1	q2
$q2^+$	q3	
q3	q4	q5
q4	q6	q5
$q5^+$	q3	
q6	q6	q5

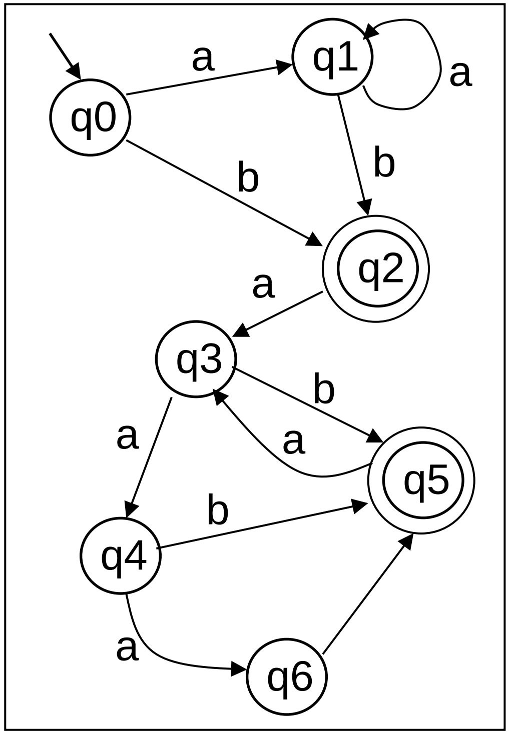


Немного не похож

Пример: Регулярное выражение $\Rightarrow$ Конечный автомат



- $R_A = \varepsilon a^*b\ (aaa^*b + ab)^*\ \varepsilon$



	δ	
	x=a	x=b
q0	q1	q2
q1	q1	q2
$q2^+$	q3	
q3	q4	q5
q4	q6	q5
$q5^+$	q3	
q6	q6	q5

	6	5^+	4	3	2^+	1
0						
1						
2^+						
3						
4						
5^+						

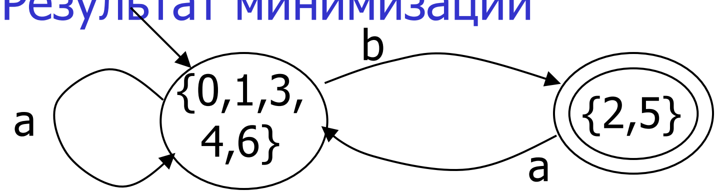
	6	5^+	4	3	2^+	1
0		N			N	
1		N			N	
2^+	N		N	N		
3		N				
4		N				
5^+	N					

Минимизируем!

$\pi 0$ = {q0, q1, q3, q4, q6}; {q2, q5}

$\pi 1$ = {q0, q1, q3, q4, q6}; {q2, q5} = $\pi 0$

Результат минимизации

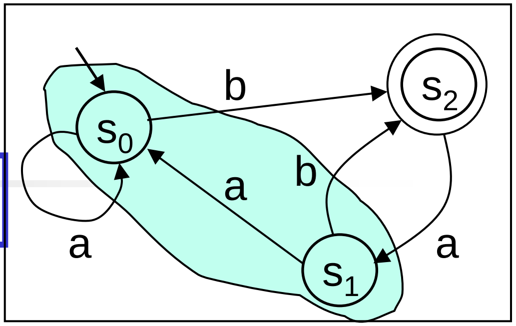


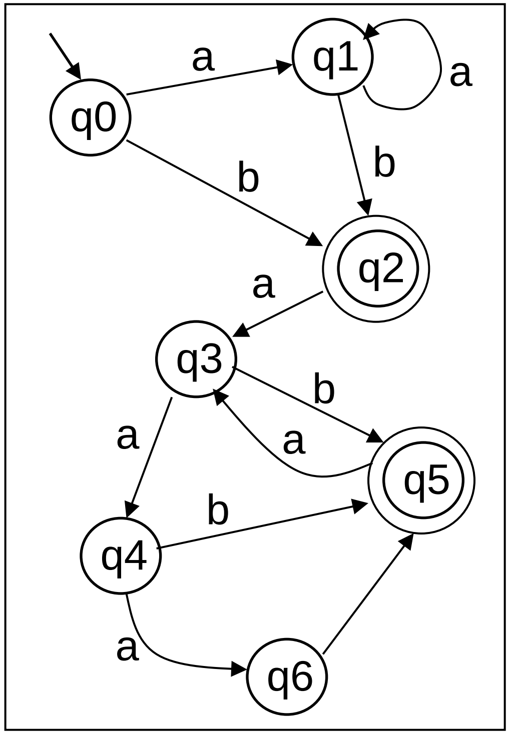
Опять не похож!!

Пример: Регулярное выражение $\Rightarrow$ Конечный автомат

- $R_A = \varepsilon a^*b\ (aaa^*b + ab)^*\ \varepsilon$

$s_0 \approx s_1$





	δ	
	x=a	x=b
q0	q1	q2
q1	q1	q2
$q2^+$	q3	
q3	q4	q5
q4	q6	q5
$q5^+$	q3	
q6	q6	q5

Исходный автомат не минимален!!

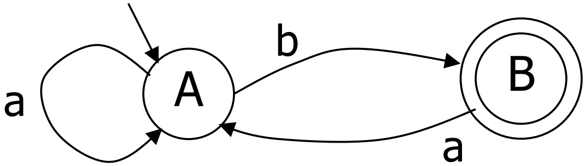
Эквивалентен исходному??

Минимизируем!

$\pi 0$ = {q0, q1, q3, q4, q6}=A; {q2, q5}=B

$\pi 1$ = {q0, q1, q3, q4, q6}; {q2, q5} = $\pi 0$

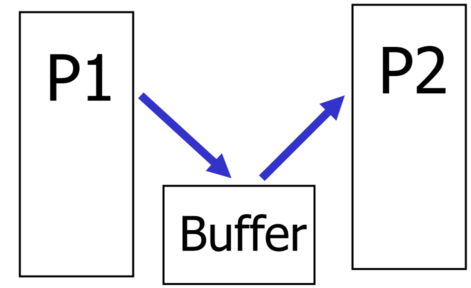
Результат минимизации



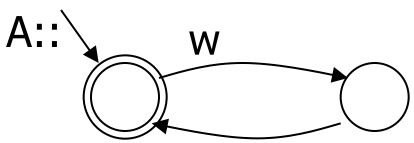
Опять не похож!!

Пример. Статический анализ текста || программ

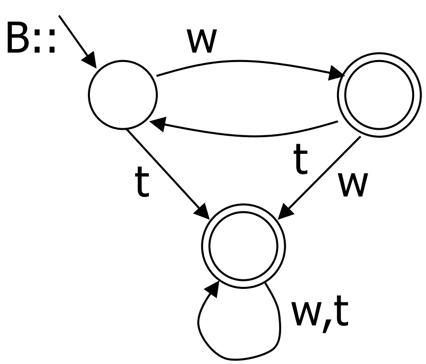
- Все возможные правильные последовательности событий обращения к ограниченному буферу (запись, w от write и выборка, t от take) представляют собой язык (wt)*
- Анализ статического кода параллельной программы, работающей с буфером, может выявить ошибку обращений к буферу (выявить любую неправильную подцепочку в потоке)



Правильные: (wt)*

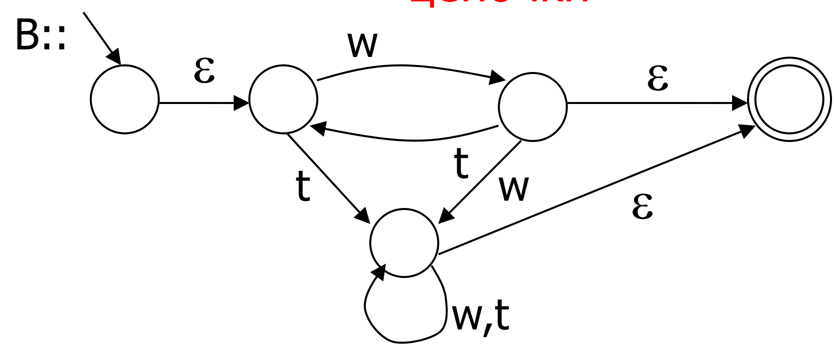


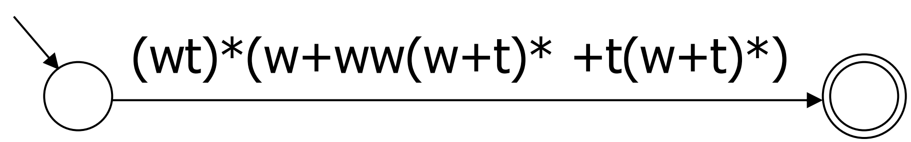
Автомат А для (wt)*



Автомат В – дополнение А

Автомат В определяет все неправильные цепочки





Неправильные: R = (wt)* (w + ww(w+t)* + t(w+t)*)

Проблема:
Эквивалентность регулярных выражений

- Теорема. Проблема эквивалентности двух регулярных выражений разрешима.
- Доказательство.

Пусть даны регулярные выражения R1 и R2. Для каждого регулярного выражения строим конечный автомат, распознающий тот язык, который задается этим регулярным выражением. Пусть для R1 и R2 соответствующие конечные автоматы A_{R1} и A_{R2}.

Для проверки эквивалентности выражений R1 и R2 достаточно проверить эквивалентность соответствующих автоматов A_{R1} и A_{R2}., а эта проблема разрешима.

Проблема: Каноническое представление регулярных выражений

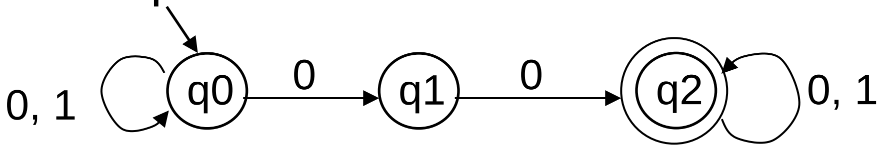
Канонического представления регулярного выражения не существует

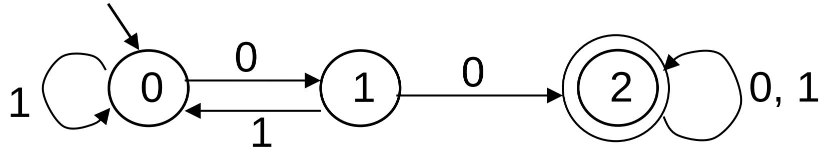
(в классе регулярных выражений)

Каноническим представлением регулярного выражения можем считать минимальный детерминированный конечный автомат, распознающий соответствующий регулярный язык

Пример

- Задача: построить регулярное выражение, описывающее множество цепочек из 0 и 1, в которых не встречаются рядом два 0
- Решение 1 – долго думать, и, наконец, выдать: $(1 + 01)^*(\varepsilon + 0)$
- Решение 2 – По регулярному выражению $(0 + 1)^*\ 00\ (0 + 1)^*$ построить автомат, построить его дополнение, и по нему построить регулярное выражение





δ	x=0	x=1
$q0^-$	{q0, q1 }	q0
q1	q2	
$q2^+$	q2	q2

Недетерминированный

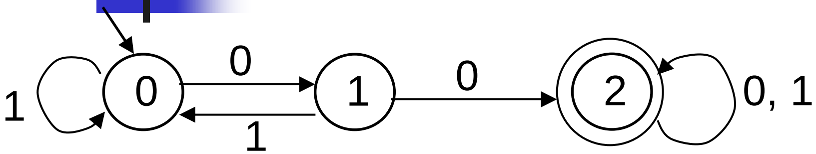
	δ	x=0	x=1
0^-	$\{ q0^- \}^-$	{q0,q1}	{ q0 }
1	{ q0, q1 }	{q0,q1,q2}	{q0}
2^+	$\{ q0,q1,q2 \}^+$	{q0,q1,q2}	{ q0,q2 }
3^+	$\{ q0, q2 \}^+$	{q0,q1,q2}	{ q0, q2 }

Детерминированный неминимальный

δ	x=0	x=1
0^-	1	0
1	2	0
2^+	2	2

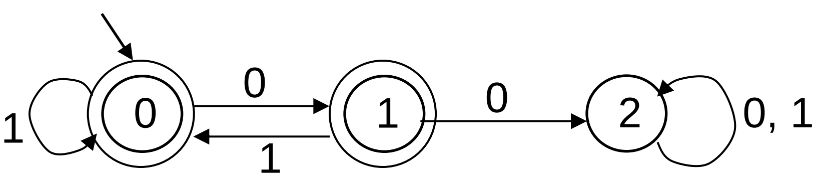
Детерминированный минимальный

Продолжение

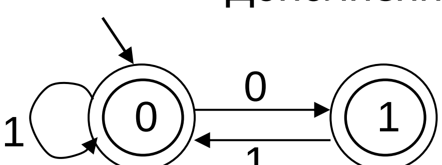


δ	x=0	x=1
0^-	1	0
1	2	0
2^+	2	2

Детерминированный минимальный

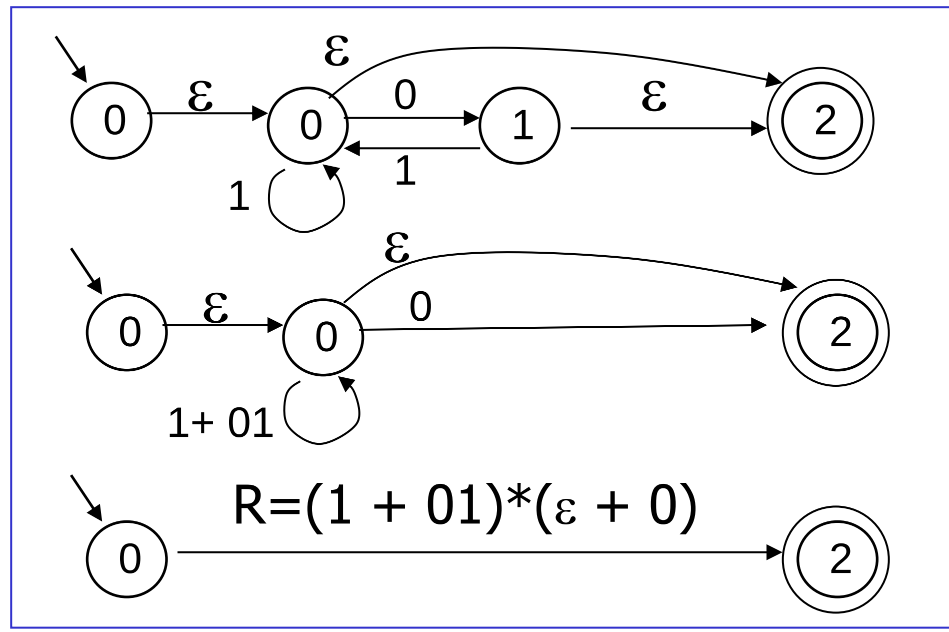


Дополнение



Выбрасываем недостижимое состояние

Строим регулярное выражение



Регулярное выражение, описывающее множество цепочек из 0 и 1, в которых нет рядом двух 0

$R=(1 + 01)^*(\varepsilon + 0)$

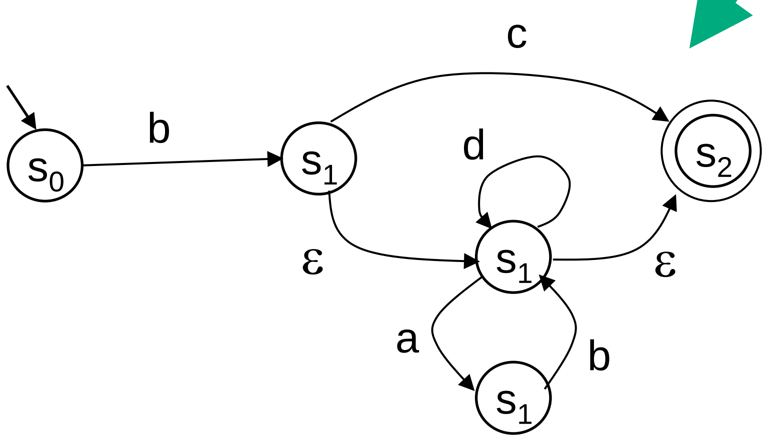
Регулярные выражения, регулярные множества и КА

$$b\ (c + (d + a\,b)^*)$$

регулярное выражение (формула R)

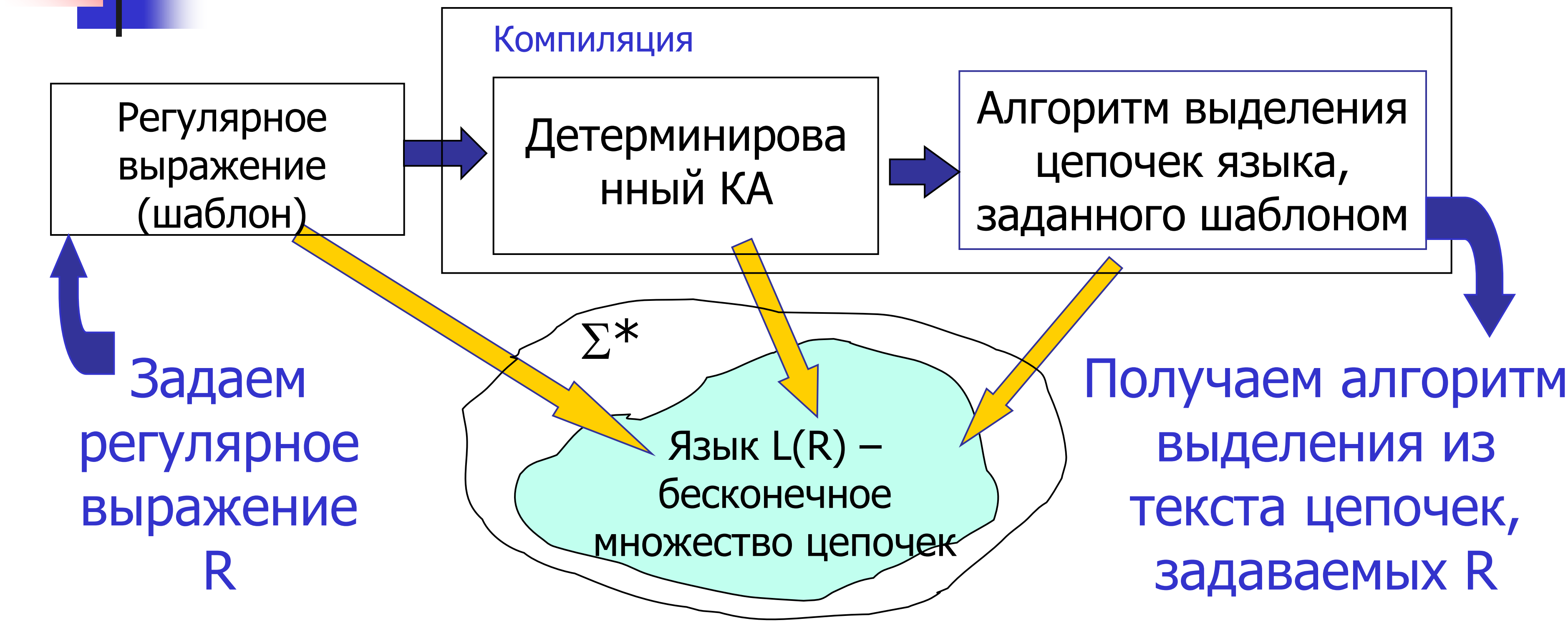
$$\{b\}\bullet(\{c\} \cup (\{d\} \cup \{a\}\bullet\{b\})^*)$$

последовательность операций над одноэлементными множествами символов, дающая регулярный язык L



конечный автомат, распознающий регулярный язык L, получающийся последовательностью операций R

Значение теоремы Клини



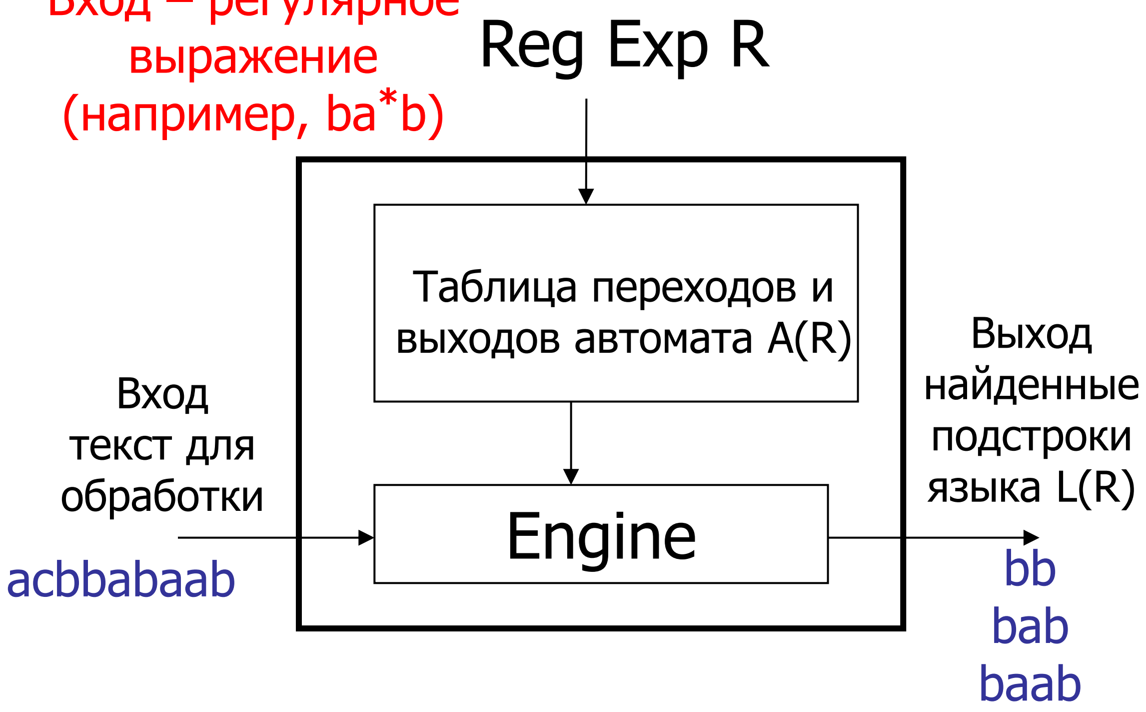
- Можно с помощью регулярного выражения (шаблона) R задать бесконечное множество цепочек L(R) , а по R построить автоматически детерминированный конечный автомат, который в потоке символов может выделить цепочки этого множества и выполнить некоторые действия – преобразования входного потока символов. Программная реализация этого детерминированного КА дает алгоритм выделения и преобразования цепочек

Программная реализация КА для обработки текстов на основе регулярных выражений

Проблема. Задано Регулярное выражение R (например, ba^*b).
Найти в тексте все цепочки, принадлежащие языку L_R

- Строим программу с настраиваемой функцией переходов и выходов
  - строится engine
  - engine работает с настраиваемой таблицей переходов и выходов КА

Вход – регулярное выражение (например, ba^*b)



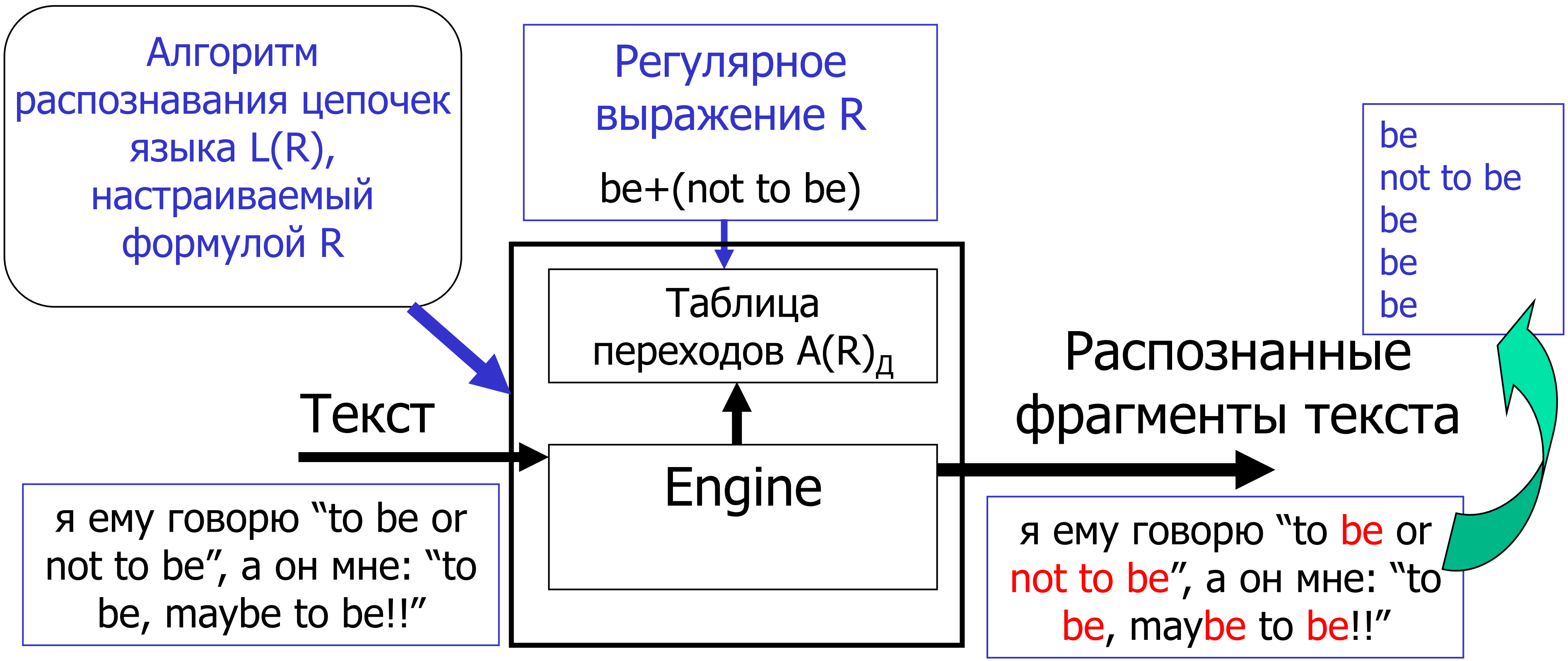
Engine работает с функцией, которая по номеру текущего состояния возвращает пару: номер следующего состояния и номер действия (функция переходов и выходов)

Таблица переходов и вых генерируется по RegExp

Engine строится один раз

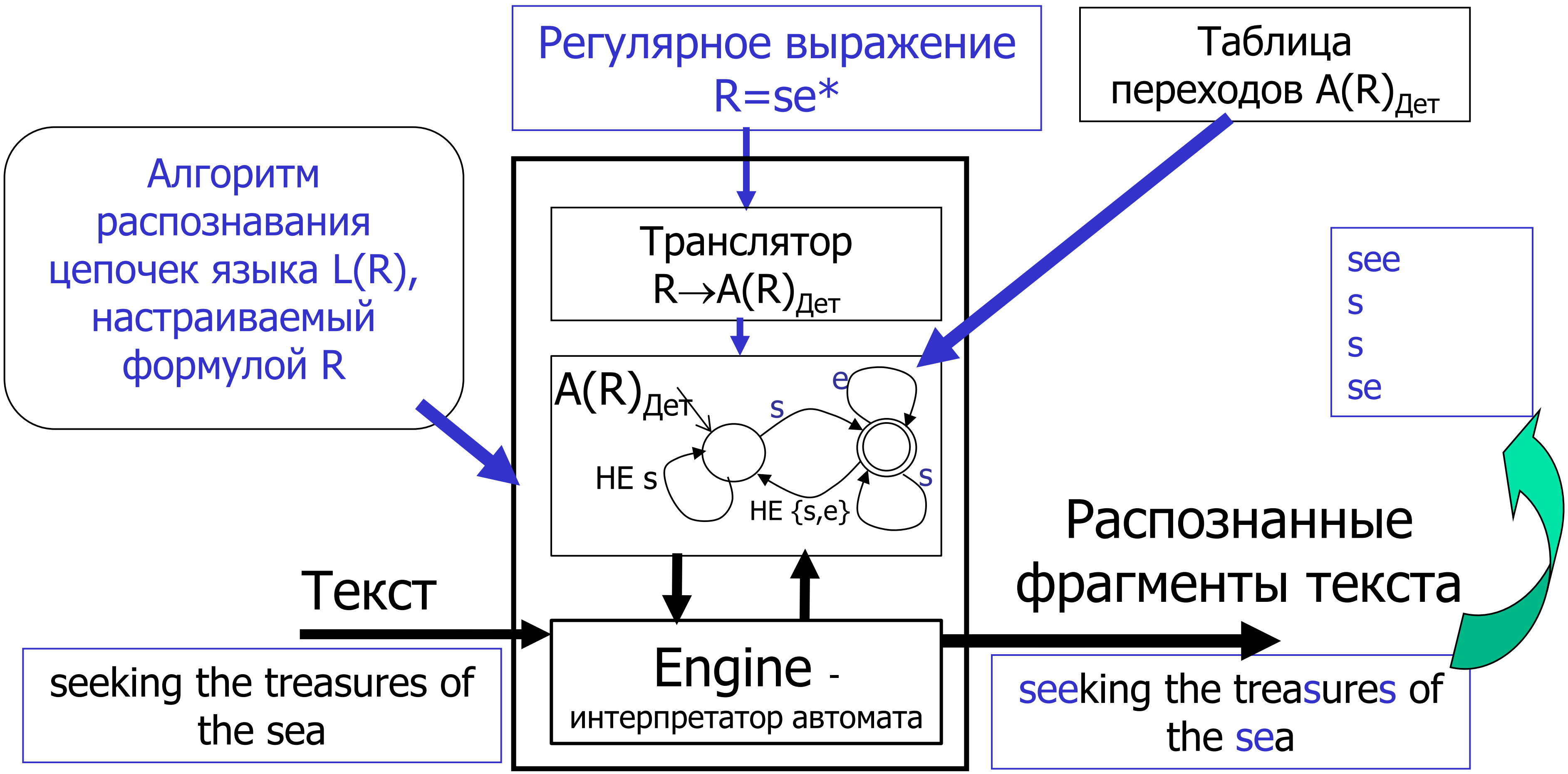
Регулярное выражение “настраивает” алгоритм распознавания

- Конечный автомат A(R) по регулярному выражению R строится автоматически
- Автомат A(R) преобразуется в детерминированный $A(R)_Д$, а по нему с помощью Engine реализуется алгоритм распознавания цепочек языка L(R)



Другое регулярное выражение – другой алгоритм

Обычно используются “жадные” алгоритмы, ‘отъедающие’ максимальные подстроки, удовлетворяющие определению



Использование регулярных выражений в практике программирования

- Как описать целые числа без знака? – просто, но громоздко!!
(0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)+ (знак '+' используется для итерации!!, т.е. 1 либо произвольное конечное число повторений)

- Как описать идентификаторы? цепочки букв и цифр, начинающиеся с буквы
(a | b | … | z | A | B | … | Z) (a | b | … | z | A | B | … | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)* - громоздко!! Более 300 символов!

- Используется 'syntactic sugar' – удобные условности и обозначения
  - Вводятся классы. [abcd] – это a|b|c|d, [P-Q] это все символы с кодировками от P до Q. Тогда:
    - целые без знака: [0-9]
    - идентификаторы: [a-zA-Z] [a-zA-Z0-9]*
  - Но как быть, если нужно в тексте распознавать скобки (,), [,], знаки *, + и т.д.?

Итак, для удобства вводятся многие условности
Также нужны правила, позволяющая специальные символы видеть в тексте как 'буквальные', а некоторые буквальные - как 'специальные', например ' ['

Классы

Целое б.з.: (0|1|2|3|4|5|6|7|8|9)+ - неудобно, слишком громоздко [0-9]+

Примеры:

[0-9]{2,} любая последовательность из не менее чем двух цифр

[a-zA-Z] [a-zA-Z0-9]* любой идентификатор: любая последовательность букв и цифр, начинающаяся с буквы

Используя квадратные скобки, альтернативу можно представить без знака '|'. Выражение st(o|au)m эквивалентно: 'st[o(au)]m'
st[oau]m задает три строки: stom, stam, stum

Классический синтаксис регулярных выражений vs конкретный синтаксис регулярных выражений

- В Языках программирования рег выражения используются очень широко
- КОНКРЕТНЫЙ СИНТАКСИС РЕГУЛЯРНЫХ ВЫР. ЗАВИСИТ ОТ РЕАЛИЗАЦИИ
- Используется ‘syntactic sugar’

1. Символ обозначает сам себя (такие символы называются литералами)

   Литералу а соответствуют три выделенные буквы а в слове ‘м**а**к**а**рон**а**’

2. Последовательно идущие литералы – это конкатенация символов

   Выражению stom соответствуют выделенные подстроки в тексте ‘cu**stom** stops **stom**atology goods’

3. Операции итерации * (повторение 0 и более раз) и усеченной итерации + (1 и более раз) (определяют ‘жадный’ алгоритм выделения)

   Выражению se* соответствуют 4 выделенные подстроки в тексте ‘**see**king trea**s**ures in **se**as’. В этом же тексте выражению se+ соответствует только две подстроки ‘**see**king treasures in **se**as’

   go+gle означает gogle, google, gooogle, goooogle, ...

Конкретный синтаксис регулярных выражений

4. Круглые скобки используются для логического выделения группы литералов. Группа обрабатывается, как единое целое

   Выражение r(sa*)* определяет следующие подстроки текста

   'Corsar sailed straight along the courses': {rsa, r, rs}

5. Метасимвол '|' (или) обозначает альтернативу – вместо '+', котрый обозначает усеченную итерацию

   Выражению st(o|au)m соответствует две подстроки: 'stom' и 'staum'

6. Приоритеты операций

   Операция '|' имеет наименьший приоритет.

   sto|aum трактуется, как две строки: sto и aum, поскольку конкатенация - более приоритетная операция, чем |

   st(o|au)m - трактуется, как две строки: stom и staum

Повторения (квантификаторы)

7. Для указания повторений предыдущего элемента кроме * и + в качестве квантификатора используются фигурные скобки и знак вопроса ?:

{n} - n вхождений

{n,} - n или более вхождений (Например, (ab){1,} эквивалентно (ab)+)

{n,m} - не менее n и не более m вхождений

? – "необязательность", 0 или 1 раз повторение предыдущего символа (группы), эквивалентно {0,1}

Пример abc{1,3}de? - 6 строк: abcd, abccd, abcccd, abcde, abccde, abcccde

8. Квадратные скобки используются для указания классов - групп символов, фактически, альтернатив из множеств символов

[xyz] любой из символов x, y или z (эквивалентно x|y|z)

[a-z] любой символ в указанном диапазоне (эквивалентно a|b|c|d|...|z)

[^xyz] любой символ, кроме тех, которые указаны далее в скобках

[^a-z] любой символ, кроме указанных в диапазоне

Summary: специальные символы (метасимволы)

( ) [ ] { } \ . | ^ * + ? $ - эти символы – служебные, вспомогательные

( ) – группировка символов и запоминание

[ ] – классы: [abc] – это a+b+c в классической записи, [a-d] означает a+b+c+d

{3} – ровно 3 повторения предыдущего символа (группы)

{3, 6} – не менее 3 и не более 6 повторений предыдущего символа (группы)

{3, } – не менее 3 повторений предыдущего символа (группы)

\ - обратный слэш для переключения метасимволов (см. дальше)

. – точка, обозначает любой один символ

| - знак объединения (вместо '+' используемого в классической записи)

e? – 0 или 1 раз повторение предыдущего элемента (группы) e
Эквивалентно e {0,1}

^ - начало строки (указание – поиск вести с начала строки), ^ после [
означает дополнение до множества перечисленных символов

$ - конец строки в тексте (указание – поиск вести до конца строки)

Примеры

.illy - любая группа из 5 символов, оканчивающаяся на illy (Billy, tilly, ...)

ко(т|шка) - кот, кошка

.* - произвольное число любых символов

‘\’ перед специальным символом

- Специальные символы иногда нужно представить буквально

( ) [ ] { } \ . | ^ * + ? $ - эти символы – служебные, вспомогательные

- Если ‘\’ стоит перед специальным символом, то этот символ - литерал:
  - \[- трактуем как открывающую квадратную скобку
  - \| - вертикальная палка. Обычно ‘|’ обозначает альтернативу
  - \^ - сам символ ‘^’. Обычно он означает начало строки или отрицание
  - \\ - обратный слэш (один!)
  - \. – точка. Обычно точка трактуется, как любой символ (в некоторых реализациях - кроме символа начала строки)
- Примеры
  - a\.? – строка ‘a.’ или ‘a’
  - a\\\\b – строка ‘a\\b’
  - a\[F\] - строка a[F]

Обратный слэш '\' перед обычными символами

- Если '\' стоит перед некоторыми литералами, которые трактуются буквально, то они становятся специальными символами (метасимволами):
  - 'Белые' символы: \f – символ перевода формата (FF)
    \n – символ перевода строки
    \r – символ возврата каретки (CR)
    \t – символ табуляции
    \v – символ вертикальной табуляции
  - \s – любой 'белый', невидимый символ, эквивалентно [\f\n\r\t\v]
  - \S – любой видимый символ
  - \w – любой буквенный символ, эквивалентно [a-zA-Z]
  - \W – все символы, кроме букв, эквивалентно [^ a-zA-Z]
  - \d - любой символ – цифра, эквивалентно [0-9]
  - \D – любой символ, кроме цифры, эквивалентно [^0-9]
  - \<, \> – отмечают, соответственно, начало и конец слова в тексте
- Примеры
  - Идентификатор: вместо [a-zA-Z] [a-zA-Z0-9]* можем писать \w[\w\d]*

Специальные ⇔ буквальные символы с помощью '\'

Специальные символы
(т.е. имеющие специальный смысл)

( ) [ ] . * + | …

\w, \s, \d, \f, …

Добавление слэша '\'

Добавление слэша '\'

Буквальные символы
(т.е. имеющие буквальный смысл: символ)

\(, \), \[, …

w, s, d, f, …

Пример:

egrep и Perl интерпретируют скобки и вертикальную черту как спецсимволы, если перед ними *нет* обратной косой черты и воспринимают их как буквальные символы, если черта есть

Обработка строк в современных языках программирования

- ^ - начало строки (если не сразу после `[`)
  $ - конец строки

- Примеры:
  - ^From – те строки, в начале которых стоят четыре буквы F, r, o, m
  - ^$ - найдет пустые строки, т.е. те строки, в которых сразу после начала идет конец

(?: aaa) – группировка без запоминания
/ - ограничитель шаблона. Шаблон всегда записывается между парой слэшей.
После последнего слэша могут идти параметры
/ <регэксп>/i – не различать прописные и заглавные буквы только если i стоит после конца ограничителя шаблона
/ <регэксп>/x – игнорировать пробелы и переводы строк– только если x стоит после конца ограничителя шаблона

Примеры

- / ([^\s]+)\s+[^\s\.]+/ - шаблон языка Perl, разберем по порядку:

  [^\s]+ - любая последовательность не ‘белых’ символов (пробелов), содержащая по крайней мере один символ

  \s+ - по меньшей мере один пробел, но может быть сколько угодно

  [^\s\.]+ - все, что не пробел и не точка, содержащее по крайней мере один символ

- [,:\?]$ - подстрока, состоящая ровно из одного символа ‘,’ , ‘:’ или ‘?’, стоящая в конце строки (знак $ означает конец строки)
- /[0-9]+\. [0-9]* | \.[0-9]+/ - десятичная вещественная константа без знака, с дробной частью или без нее, как шаблон (между двумя слэшами)
- /\d+\.\d*|\.\d+/ - то же

Пример

- /^[a-zA-Z0-9]+$/

[a-zA-Z0-9]	прописная или строчная буква, либо цифра
[a-zA-Z0-9]+	много, но не менее одного такого символа
^[a-zA-Z0-9]+$	искать от начала строки до конца строки
/^[a-zA-Z0-9]+$/	это шаблон языка Perl (в слэшах)

Пример: анализ сложного регулярного выражения

/^((8|\+7) [\-]?)? (\(?\d{3}\)? [\-]?)? [\d\-]{7,10}$/ - ЧТО ЭТО?

/^ ...... $/	^ - начало строки, $ - конец строки, / ... / - шаблон
(8|\+7)	текст начинается либо с 8, либо +7
[\-]?	затем идет необязательные символ '-' или пробел
((8|\+7)[\-]?)**?**	вся эта группа может и не встретиться
\(?\d{3}\)?	далее идет группа из трех цифр в скобках причем скобок может и не быть
[\-]?	необязательное тире либо пробел
(\(?\d{3}\)? [\-]?)?	вся группа необязательна
[\d\-]{7,10}	от 7 до 10 цифр, либо тире либо пробелов

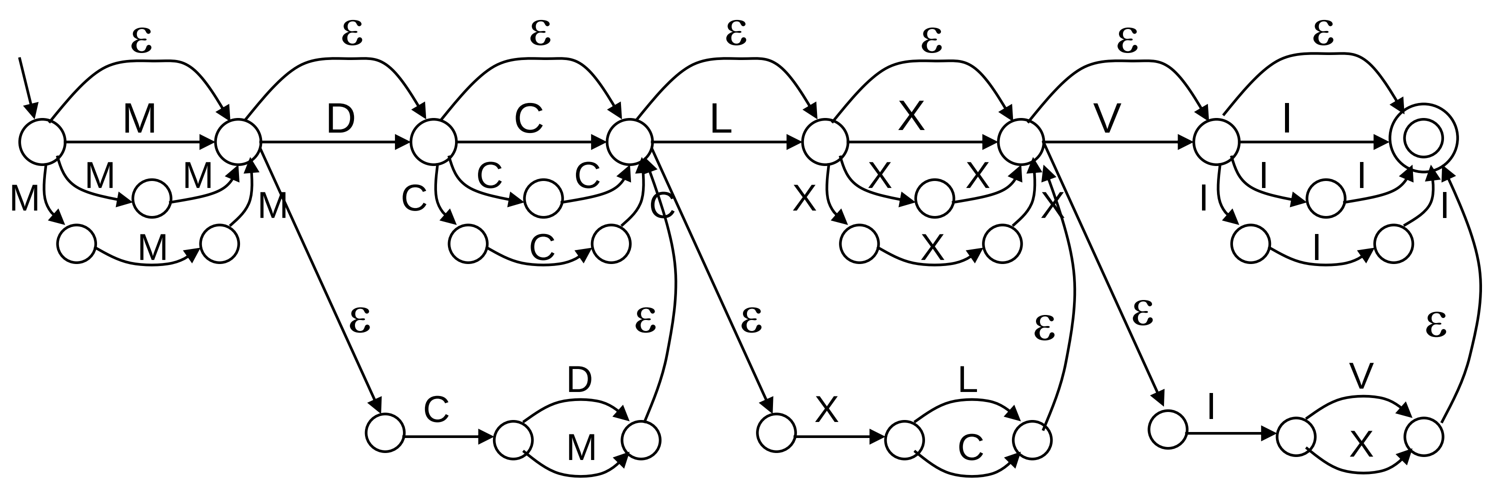
ЭТО – шаблон тел. номера России, но может быть и 'шум'. Например, строка +7 -22-3 3--4. Не всегда просто подобрать точный шаблон. Иногда невозможно

Общее замечание: нет единого стандарта конкретного синтаксиса (syntactic sugar) представления регулярных выражений в современных программных системах. Обычно требуется изучение документации для выявления нюансов

Регулярное выражение и НДКА для римских чисел

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Недетерминированный конечный автомат, описывающий язык римских чисел – вместо сложных правил



Регулярное выражение, описывающее весь язык римских чисел – вместо сложных правил

M{0,3} (D? C{0,3} | C[DM])? (L? X{0,3} | X[LC])? (V? I{0,3} | I [VX])?

тысячи сотни десятки единицы

Возможная курсовая работа: настраиваемый автомат, распознающий регулярные языки

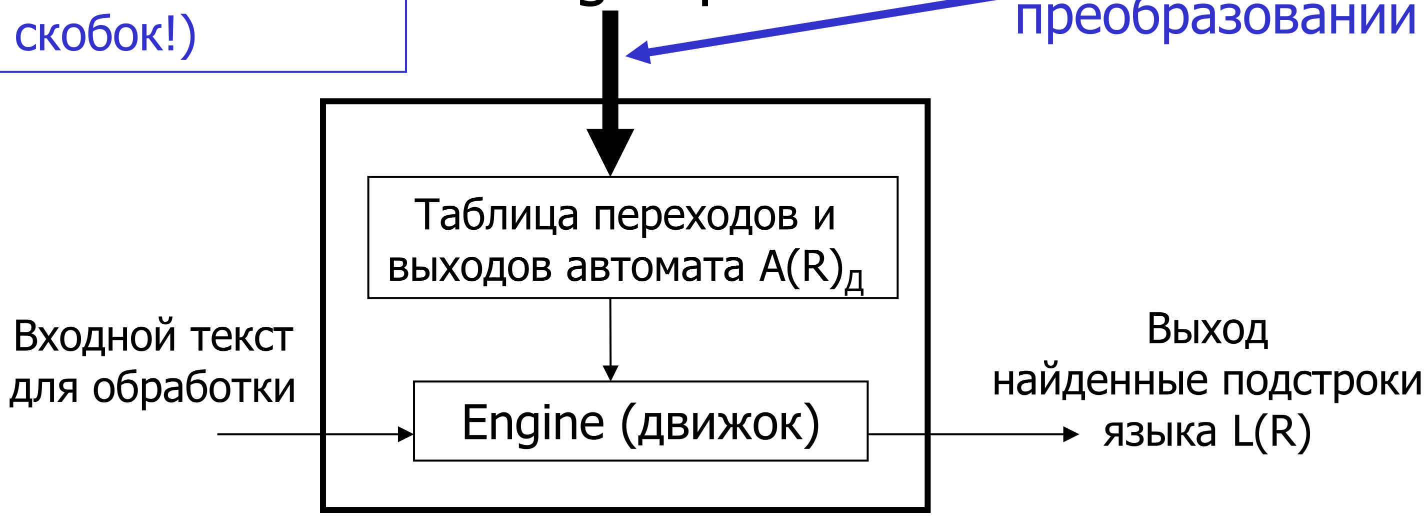
- Построить алгоритм, который по записи произвольного регулярного выражения строит автомат, распознающий нужные подстроки
- Фактически, проблема состоит в том, чтобы построить компилятор компиляторов, т.е. транслятор, который по записи R регулярного выражения строил бы таблицу переходов и выходов автомата $A(R)_Д$

/ ([^\s]+)\s+[^\s\.]+/

Язык Рег выр – не автоматный (вложенности скобок!)

Reg Exp R

Речь идет об этом преобразовании

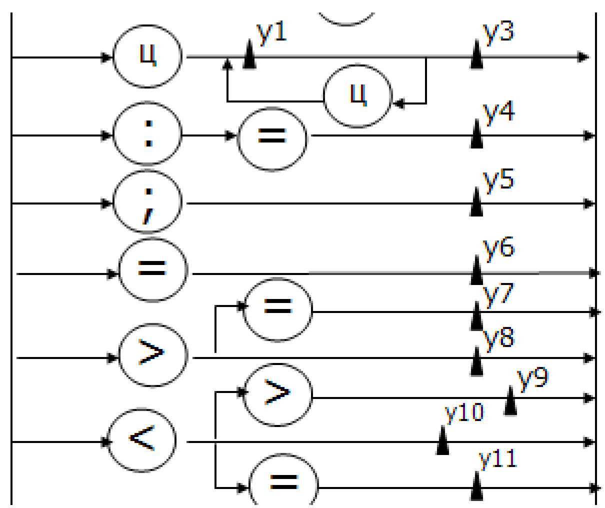


Возможная курсовая работа: настраиваемый лексический анализатор

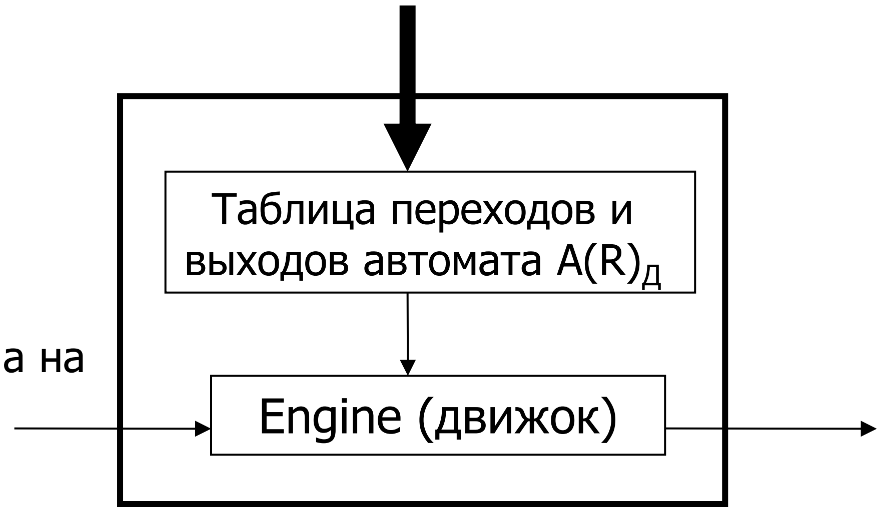
- Лексический анализатор, настраиваемый для любого конкретного языка

Входной файл:
описание лексем языка шаблонами и семантики - что делать после распознавания каждой лексемы

Фрагмент сканера



Вход:
программа на языке ВУ

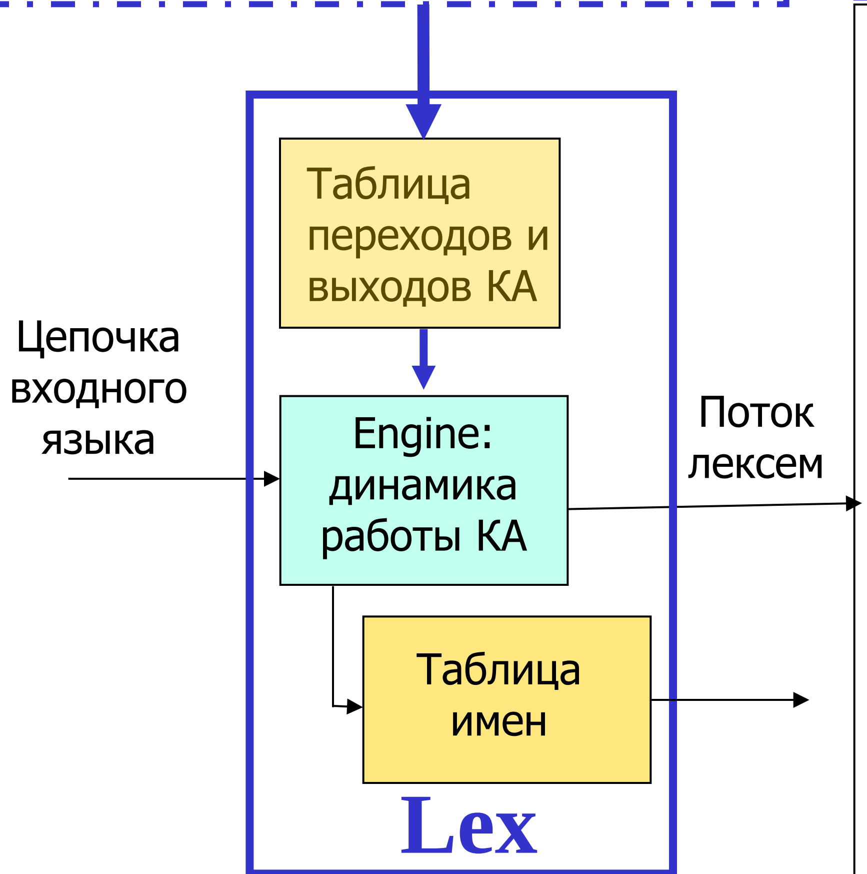


Выход:
программа в виде потока лексем
+ таблица имен
+ таблица констант

Лексический анализатор Lex (1975 г.)

Входной файл Lex:

1) Регулярные выражения для лексем
2) Семантика для них на Си



Lex читает входной файл и выдаёт код лексического анализатора на Си: Таблицу переходов и выходов КА и Engine, который работает по этой Таблице

Структура входного файла - три блока, разделенных %%:

Блок определений:
макросы и заголовочные файлы (может отсутствовать)

%%

Блок правил:
шаблоны (рег выр), и оператор или вызов функции на Си, которые должны выполняться для каждой подстроки входного файла при совпадении ее с шаблоном

%%

Блок кода:
операторы и функции на Си, вызываемые из Блока правил для обработки цепочек, выделенных по шаблонам из входного потока (могут отсутствовать)

Заключение

- Регулярные множества – (бесконечные) множества цепочек, получаемые из символов словаря любой конечной последовательностью трех операций: объединение, конкатенация, итерация
- Регулярные выражения – формулы, показывающие порядок применения операций к одноэлементным множествам. они конечным образом описывают регулярные языки
- Теорема Клини: классы регулярных множеств и автоматных языков совпадают
- Следствием теоремы Клини является то, что по конечному автомату, задающему автоматный язык, можно построить регулярное выражение, задающее этот язык, и обратно, по формуле – регулярному выражению, задающей регулярный язык, можно построить детерминированный конечный автомат, который распознает все цепочки этого языка (а, значит, построить алгоритм распознавания этих цепочек и их преобразования)

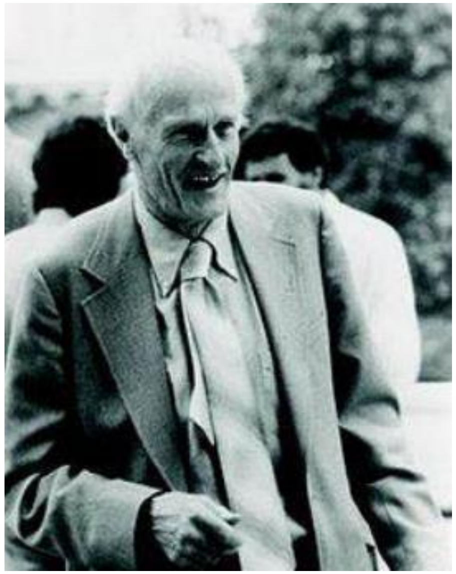
Заключение (2)

- Рассмотрена теоретическая основа регулярных выражений (англ. «regular expressions», жаргон: «регэкспы» или «регэксы»)
- Регулярные выражения являются современным мощным языком построения шаблонов (образцов) для поиска текстовых фрагментов в электронных документах
- Во всех современных языках существуют классы RegExp, методы которых основывается на описанных здесь алгоритмах и идеях. Регулярные выражения произвели прорыв в электронной обработке текстов в конце XX века
- Не всегда легко разобраться в сложной конструкции регулярного выражения, потому что в него включается и дополнительная информация, например, метасимволы и семантика (что делать с выбранными фрагментами)
- Во многих реализациях стандарты представления Регулярных Выражений и семантики отличаются, НО СУТЬ ОСТАЕТСЯ ТОЙ ЖЕ: описание автоматного языка с помощью формул – регулярных выражений, и автоматическое построение программы-распознавателя и преобразователя входных текстов

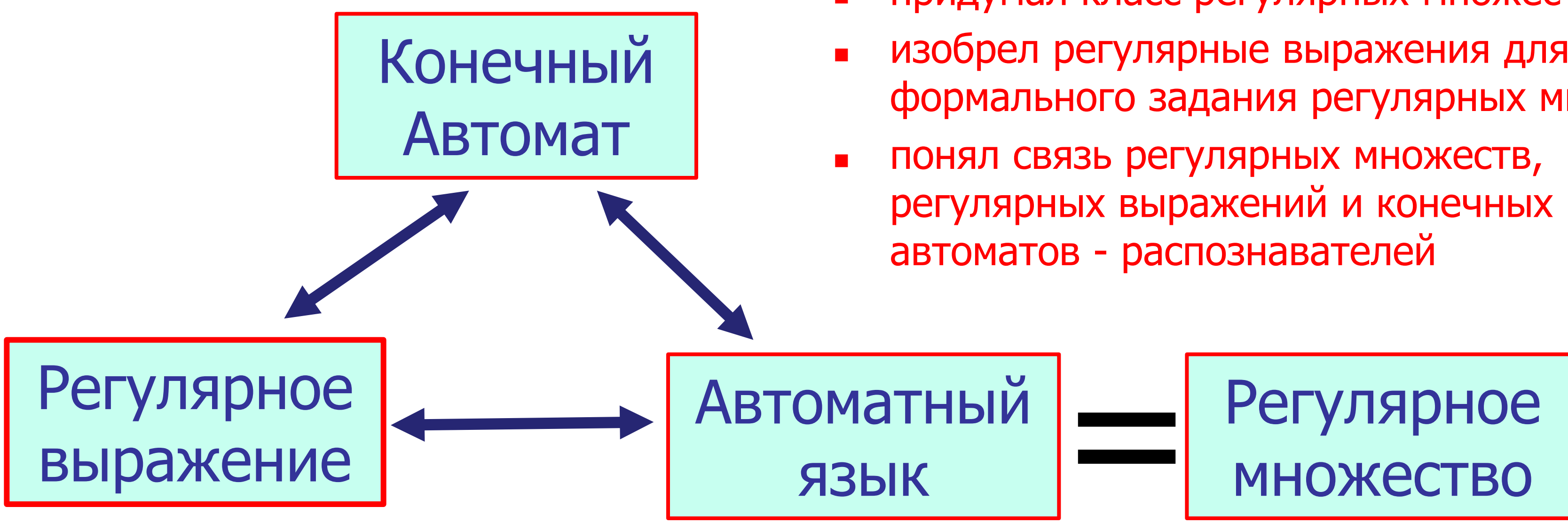
Применять регулярные выражения или нет?

- Ответ на этот вопрос зависит не только от задачи, но и от наличия опыта
- Регулярные выражения применяют для поиска в потоке подстроки, удовлетворяющей шаблону
- С каждым шаблоном можно связать алгоритм обработки тех строк, которые во входном потоке выявляются автоматически по описанию шаблона – замена подстроки, изменение формата, удаление, составление журнала, подсчет ...
- Регулярные выражения — не самое простое из того, что есть в программировании. Не всегда легко разобраться в сложной конструкции. Существуют разные стандарты и диалекты регулярных выражений
- Но есть задачи, которые можно решить с помощью регулярных выражений на порядок быстрее, чем другими методами

Персоналии: Стефен Клини



- Стефен Клини (Stephen Cole Kleene) (1909 – 1994) – выдающийся американский математик и логик, внесший огромный вклад в логику и теоретическую информатику
- Фундаментальные работы в логике, вместе с Эмилем Постом и Аланом Тьюрингом заложил основы теории вычислимости
- Кроме того:
  - придумал класс регулярных множеств
  - изобрел регулярные выражения для формального задания регулярных множеств
  - понял связь регулярных множеств, регулярных выражений и конечных автоматов - распознавателей



Источник и справочник

- Джеффри Фридл. «Regular expressions. Регулярные выражения», издана на многих языках, несколько изданий, третье на русском - 2008 г.

- Регулярные выражения обладают исключительно богатыми возможностями по обработке текстов

Требования к знаниям студента по разделам курса

Конечные автоматы

Грамматики Хомского

Атрибутная семантика

Синтаксический анализ

Трансляция ЯВУ

- понимать связь абстрактных автоматов и проблем теории формальных языков, уметь определять формальные языки;
- знать определение грамматики, различия между порождающими и распознающими грамматиками
- знать определение автоматных языков и конечных автоматов как моделей задания формальных языков
- уметь проверять эквивалентность двух автоматов-распознавателей, выполнять минимизацию автоматов-распознавателей, строить синхронную и асинхронную композицию конечных автоматов
- понимать и уметь доказывать теоремы Мура, Рабина-Скотта, Майхилла-Нероуда, Клини
- понимать идею модели недетерминированного конечного автомата-распознавателя, уметь строить эквивалентный детерминированный КА для недетерминированного КА
- понимать связь КА и синтаксических диаграмм
- уметь строить трансляторы для автоматных языков, лексические анализаторы для ЯВУ
- понимать связь и различие регулярных множеств, регулярных выражений, КА и автоматных грамматик
- уметь использовать теорему Клини для построения РВ⇔КА
- уметь использовать возможности регулярных выражений и технологию работы с ними для задач обработки текстов

Спасибо за внимание