**Контрольная работа для заочной формы магистрантов**

**8 вариант**

 **- выполнить задание №1 (6 задач из 27)**

 **- выполнить задание №2 «Исследование регулярных языков и автоматов» по методичке.**

 Задание №1

 Выбор вариантов выполняемых вопросов, вариантов должно быть 6.

 Вопросов всего 27. Разбить их последовательно на 9 троек: (1,2,3), (4,5,6),…(25,26,27).

 Каждому варианту будут соответствовать две различные тройки согласно ниже представленной таблице.

 Таблица выбора варианта

|  |  |
| --- | --- |
| Тройки вопросов |  Варианты по номеру фамилии в журнале |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | ***8*** | 9 | 10 | 11 | 12 | 13 | 14 |
| 1 | • |  |  |  | • |  |  |  |  |  |  | • |  |  |
| 2 |  | • |  |  |  | • |  |  |  |  |  |  | • |  |
| 3 |  |  | • |  | • |  | • |  |  |  |  |  |  | • |
| 4 |  |  |  | • |  | • |  | ***•*** |  |  |  | • |  |  |
| 5 |  |  |  |  |  |  | • |  | • |  |  |  | • |  |
| 6 |  |  |  | • |  |  |  | ***•*** |  | • |  |  |  | • |
| 7 |  |  | • |  |  |  |  |  | • |  | • |  |  |  |
| 8 |  | • |  |  |  |  |  |  |  | • |  |  |  |  |
| 9 | • |  |  |  |  |  |  |  |  |  | • |  |  |  |

1. Дана грамматика. Построить вывод заданной цепочки.

 a) S  T | T+S | T-S b) S  aSBC | abC

 T  F | F\*T CB  BC

 F  a | b bB  bb

 Цепочка a-b\*a+b bC  bc

 cC  cc

 Цепочка aaabbbccc

2. Построить все сентенциальные формы для грамматики с правилами:

 S  A+B | B+A

 A  a

 B  b

3. К какому типу по Хомскому относится данная грамматика? Какой язык она порождает? Каков тип языка?

 a) S  APA b) S  aQb | ε

 P  + | - Q  cSc

 A  a | b

 c) S  1B d) S  A | SA | SB

 B  B0 | 1 A  a

 B  b

1. Построить грамматику, порождающую язык:
2. L = { an bm | n, m >= 1}
3. L = { αcβcγc | α, β, γ - любые цепочки из a и b}
4. L = { a1 a2 ... an an ... a2 a1 | ai = 0 или 1, n >= 1}
5. L = { an bm | n ≠ m ; n, m >= 0}
6. L = { цепочки из 0 и 1 с неравным числом 0 и 1}
7. L = { αα | α ∈ {a,b}+}
8. L = { ω | ω ∈ {0,1}+ и содержит равное количество 0 и 1, причем любая подцепочка, взятая с левого конца, содержит единиц не меньше, чем нулей}.
9. L = { (a2m bm)n | m >= 1, n >= 0}
10. L = { ⊥ | n >= 1}
11. L = {  | n >= 1}
12. L = { | n >= 1}

5. К какому типу по Хомскому относится данная грамматика? Какой язык она порождает? Каков тип языка?

 a) S  a | Ba b) S  Ab

 B  Bb | b A  Aa | ba

 c) S  0A1 | 01 d) S  AB

 0A  00A1 AB  BA

 A  01 A  a

 B  b

 e) S  A | B f) S  0A | 1S

 A → aAb | 0 A → 0A | 1B

 B → aBbb | 1 B → 0B | 1B | ⊥

 g) S → 0S | S0 | D h) S → 0A | 1S | ε

 D → DD | 1A | ε A → 1A | 0B

 A → 0B | ε B → 0S | 1B

 B → 0A | 0

 i) S → SS | A j) S → AB⊥

 A → a | bb A → a | cA

 B → bA

 k) S → aBA | ε l) S → Ab | c

 B → bSA A → Ba

 AA → c B → cS

6. Эквивалентны ли грамматики с правилами :

 а) S  AB и S  AS | SB | AB

 A  a | Aa A  a

 B  b | Bb B  b

 b) S  aSL | aL и S  aSBc | abc

 L  Kc cB  Bc

 cK  Kc bB  bb

 K  b

7. Построить КС-грамматику, эквивалентную грамматике с правилами:

 а) S  aAb b) S  AB | ABS

 aA  aaAb AB  BA

 A  ε BA  AB

 A  a

 B  b

8. Построить регулярную грамматику, эквивалентную грамматике с правилами:

 а) S  A | AS b) S  A**.**A

 A  a | bb A  B | BA

 B  0 | 1

9. Доказать, что грамматика с правилами:

 S  aSBC | abC

 CB  BC

 bB  bb

 bC  bc

 cC  cc

порождает язык L = {an bn cn | n >= 1}. Для этого показать, что в данной грамматике

1. выводится любая цепочка вида an bn cn (n >= 1) и
2. не выводятся никакие другие цепочки.

10. Дана грамматика с правилами:

 a) S  S0 | S1 | D0 | D1 b) S  if B then S | B = E

 D  H. E  B | B + E

 H  0 | 1 | H0 | H1 B  a | b

Построить восходящим и нисходящим методами дерево вывода для цепочки:

a) 10.1001 b) if a then b = a+b+b

11. Определить тип грамматики. Описать язык, порождаемый этой грамматикой. Написать для этого языка КС-грамматику.

 S  P⊥

 P  1P1 | 0P0 | T

 T  021 | 120R

 R1  0R

 R0  1

 R⊥ 1⊥

12. Построить регулярную грамматику, порождающую цепочки в алфавите
{a, b}, в которых символ a не встречается два раза подряд.

13. Написать КС-грамматику для языка L, построить дерево вывода и левосторонний вывод для цепочки aabbbcccc.

 L = {a2n bm c2k | m=n+k, m>1}.

14. Построить грамматику, порождающую сбалансированные относительно круглых скобок цепочки в алфавите { a, ( , ), ⊥ }. Сбалансированную цепочку α определим рекуррентно: цепочка α сбалансирована, если

1. α не содержит скобок,
2. α = (α1) или α= α1 α2, где α1 и α2 сбалансированы.

15. Написать КС-грамматику, порождающую язык L, и вывод для цепочки aacbbbcaa в этой грамматике.

 L = {an cbm can | n, m>0}.

16. Написать КС-грамматику, порождающую язык L, и вывод для цепочки 110000111 в этой грамматике.

 L = {1n 0m 1p | n+p>m; n, p, m>0}.

17. Дана грамматика G. Определить ее тип; язык, порождаемый этой грамматикой; тип языка.

 G: S  0A1

 0A  00A1

 A  ε

18. Дан язык L = {13n+2 0n | n>=0}. Определить его тип, написать грамматику, порождающую L. Построить левосторонний и правосторонний выводы, дерево разбора для цепочки 1111111100.

19. Привести пример грамматики, все правила которой имеют вид
A  Bt, либо A  tB, либо A  t, для которой не существует эквивалентной регулярной грамматики.

20. Написать общие алгоритмы построения по данным КС-грамматикам G1 и G2, порождающим языки L1 и L2, КС-грамматики для

1. L1∪L2
2. L1 \* L2
3. L1\*

**Замечание:** L =L1 \* L2 - это конкатенация языков L1 и L2, т.е.L = { αβ | α ∈ L1, β ∈ L2}; L = L1\* - это итерация языка L1, т.е. объединение {ε} ∪ L1 ∪ L1\*L1 ∪ L1\*L1\*L1 ∪ ...

21. Написать КС-грамматику для L={ωi 2 ωi+1R | i ∈ N, ωi=(i)2 - двоичное представление числа i, ωR - обращение цепочки ω}. Написать КС-грамматику для языка L\* (см. задачу 20).

22. Показать, что грамматика

 E  E+E | E\*E | (E) | i

неоднозначна. Как описать этот же язык с помощью однозначной грамматики?

23. Показать, что наличие в КС-грамматике правил вида

1. A  AA | α
2. A  AαA | β
3. A  αA | Aβ | γ

где α, β, γ ∈ (VT∪VN)\*, A ∈ VN, делает ее неоднозначной. Можно ли преобразовать эти правила таким образом, чтобы полученная эквивалентная грамматика была однозначной?

24. Показать, что грамматика G неоднозначна. Какой язык она порождает? Является ли этот язык однозначным?

 G: S  aAc | aB

 B  bc

 A → b

25. Дана КС-грамматика G={VT, VN, P, S}. Предложить алгоритм построения множества

 X={A ∈ VN | A ⇒ ε}.

26. Для произвольной КС-грамматики G предложить алгоритм, определяющий, пуст ли язык L(G).

27. Написать приведенную грамматику, эквивалентную данной.

 a) S  aABS | bCACd b) S  aAB | E

 A  bAB | cSA | cCC A  dDA | ε

 B  bAB | cSB B  bE | f

 C  cS | c C  cAB | dSD | a

 D  eA

 E  fA | g

 Задание №2

 Задание выполняется по методичке «Исследование регулярных языков и автоматов»

Государственное образовательное учреждение

высшего образования

Донской государственный технический университет

ДГТУ

Кафедра «Вычислительные системы и информационная безопасность»

 ИССЛЕДОВАНИЕ РЕГУЛЯРНЫХ ЯЗЫКОВ И АВТОМАТОВ

Методические указания к лабораторным работам

и по дисциплине «Теория формальных грамматик и автоматов»

Ростов-на-Дону

2016

Составитель: доктор технических наук профессор В.А. Фатхи

УДК 519.713

ИССЛЕДОВАНИЕ РЕГУЛЯРНЫХ ЯЗЫКОВ И АВТОМАТОВ: Метод. указания к контрольной работе по дисциплине «Теория формальных грамматик и автоматов» / ДГТУ, Ростов н/Д, 2016. — 18 с.

Приведены основные теоретические положения по регулярным языкам и конечным автоматам. Представлены алгоритмы построения регулярных языков по модели автомата и наоборот. Рассмотрены алгоритмы детерминизации и минимизации автоматов.

Печатается по решению редакционно-издательского совета академии

*Рецензент* кандидат технических наук, доцент А.Р.Айдинян

*Научный редактор* кандидат технических наук, профессор

 А.И. Зотов

 Содержание

 Введение…………………………………………………………………

1. Основные определения………………………………………………
2. Понятие регулярного выражения……………………………………
3. Детерминированные конечные автоматы…………………………..
4. Минимизация конечных автоматов…………………………………
5. Недетерминированные конечные автоматы………………………..
6. ε-автомат………………………………………………………………
7. Детерминизация недетерминированных автоматов……………….

# Имитация функционирования автоматов………………………….

1. Программное средство имитации функционирования автоматов..
2. Преобразование регулярного выражения в конечный автомат…..

11.Преобразование конечного автомата в регулярное выражение….

 12. Выполнение заданий……………………………………………….

 Литература……………………………………………………………..

 Введение

 В математической лингвистике значительное внимание уделено конечным автоматам и регулярным выражениям. Последние, являются формальными моделями разрабатываемых в радиоэлектронике устройств и преобразуемых ими последовательностей сигналов. Использование закономерностей, выявляемых в моделях, позволяет формулировать и решать различные задачи связанные, например, с решением задач оптимизации, повышения надежности, достижения отказоустойчивости.

 Последнее время и в программировании находит широкое использование так называемый автоматный подход, позволяющий

 При изучении предмета математической лингвистики существенную роль играет возможность проведения преобразований регулярных выражений в конечные автоматы и наоборот. Значительный объем графической работы при изучении может привести к ошибкам, которые не позволяют привести после проведения вычислений к исходным данным, использованным в начале работы.

 **1. Основные определения**

Алфавитом называется любое конечное множество некоторых символов.

Как правило, алфавит обозначается заглавной греческой буквой, например,

Σ = {0, 1}. В данном случае алфавит Σ содержит символы 0 и 1. Из алфавитных символов можно составлять **строки.** Строкой над алфавитом **Σ,** является, например, строка символов 1010100010. Строкой над алфавитом латинских букв является, например, строка *fabbssgf*. Пустая строка, т. е. строка, не содержащая символов, обозначается буквой ε. Длина такой строки равна нулю.

 Операция конкатенации**,** это операция над строками, когда одна строка приписывается в хвост другой. Так, если *a* = *acdb*, а b = *fge*, то *ab* = *acdbfge*.

 Множество всех строк алфавита Σ имеет специальное обозначение: Σ\*. Так, если Σ = {0, 1}, то Σ\*={ε, 10, 0, 000, 1010, 01110,…}. В любом алфавите можно составить бесконечное множество строк. Если выбрать не все строки из бесконечного множества строк Σ\*, а лишь некоторые строки,то получим так называемый, формальный язык, обозначаемый через *L*.

 Формальный язык– это любое подмножество строк в рассматриваемом алфавите. Например, из всех возможных строк алфавита, состоящего из цифр, точки и знака минуса, можно выбрать те, которые являются записями вещественных чисел: *L* = {0, -1.9, 23.456, -4.11}.

1. **Понятие регулярного выражения**

 При работе с любым конкретным языком возникает задача его формального описания. В качестве популярного инструмента, используемого для формального описания языков, используются регулярные выражения. Регулярное выражение является шаблоном, описывающим целое семейство (возможно бесконечное) строк.

 Пусть требуется описать некоторый язык над алфавитом Σ. Регулярными выражениями над Σ являются любые элементы алфавита Σ, а также, пустая строка ε:

* *a*, где *a* – любой элемент алфавита Σ;
* ε.

Иногда к регулярным выражениям добавляют еще пустое множество Ø.

К регулярным выражениям относятся и результаты следующих операций с регулярными выражениями над алфавитом Σ:

(*a* ∪ *b*), (*ab*) и *a*\*, где *a* и *b* – любые регулярные выражения над Σ.

Примеры регулярных выражений над алфавитом Σ={*a*, *b*, *c*}: ((*ac*)*a*)\*,

((((*a b*) ∪ *c*) *b)*\**c*)\*, ((*a* ∪ *b*) (*b* ∪ *c*) *a*).

 Язык, описываемый регулярным выражением *r*, обозначается L(*r*).

 Регулярному выражению вида (*a* ∪ *b*) соответствуют все строки, задаваемые выражением *a* и все строки, задаваемые выражением *b*. Если выражению *a* соответствуют строки *a*, *ab*, *aab*, а выражению *b*  – строки *baab* и *bbbb*, то выражение (*a* ∪ *b*) опишет язык, состоящий из всех перечисленных пяти строк.

 Регулярному выражению (*a b*)соответствует язык, состоящий из строк вида *xy*, где *x* – некоторая строка из языка *L*(*a*), а *y* – строка из языка *L*(*b*). То есть в язык L((*ab*)) входят все строки, которые можно получить путем дописывания в хвост некоторой строке из языка *L*(*a*) строки из языка *L*(*b*). То есть L((*a b*)) = {*abaab, abbbb*, *abbaab*, *abbbbb*, *aaabbaab*, *aaabbbbb*}.

 Регулярному выражению *a*\* соответствует язык, состоящий из строк ε, *s*1, *s*1 *s*2, *s*1 *s*2 *s*3, …, где *s*1 – любая строка из языка *L*(*a*). Операция, в результате которой возникает указанный язык, называется операцией Клини. Она определяется еще и так: *L*(*a*\*) = {ε, *a*,( *a a*), ((*a a*) *a*), (((*a a*) *a*) *a*), …}, то есть, нуль или более повторений *a*. Например, если *L*(*a*) = {*a*, *ab*, *bbbb*}, то *L*(*a*\*) = { ε, *a*, *aaaaab*, *ababbbbb*, …} – любые комбинации, составленные из произвольного числа строк.

 Необходимо отметить, что при составлении сложных регулярных выражений возникает множество круглых скобок. Чтобы этот факт не перегружал получаемые выражения, введен приоритет операций для сокращения количества скобок. Приоритет определяет выполнение в первую очередь операции замыкание Клини, затем конкатенации и, в последнюю очередь, операции объединения. Дальнейшему сокращению количества скобок приводит учет закона ассоциативности операций конкатенации и объединения, например, *a*(*b*(*c d*)) можно записать в виде *abcd*.

 Любой язык, который можно описать регулярным выражением, является регулярным. Например, любой язык, состоящий из конечного числа строк, является регулярным. Так, язык *L =* {*abc*, *aaa*, *bab*, *c*} описывается регулярным выражением *abc* ∪ *aaa* ∪ *bab* ∪ *c*, поэтому он является регулярным языком.

**3. Детерминированные конечные автоматы**

 Конечные автоматы в теоретических работах используются в качестве средства описания формальных языков, а на практике они представляют мощный инструмент разработки простых и надежных программ.

 Первым в ряду конечных автоматов является детерминированный конечный автомат.

 Для уяснения структуры конечного автомата удобно изобразить его в виде графа, содержащего узлы, которыми задаются состояния конечного автомата и ориентированные ребра, задающие правила перехода из состояния в состояние конечного автомата. В графе помечаются начальное и конечное состояния. Пример представления конечного автомата в виде графа представлен на рис. 1. Здесь 3 – начальное состояние, а 2 – конечное состояние или, так называемое, допускающее состояние.

*C*

*A*

*A*

*B*

*B*

Рис.1. Представление конечного автомата с помощью графа

 Говорят, что автомат допускает строку, если эта строка переводит его в одно из допускающих состояний. В противном случае говорят, что автомат отвергает строку. Автомат заканчивает работу в допускающем состоянии тогда и только тогда, когда на вход автомата поступает цепочка символов соответствующая регулярному выражению.

 Рассмотрим детерминированный конечный автомат на формальном уровне.

 В детерминированном конечном автомате выделяют и формально записывают следующие элементы:

* конечное множество состояний *Q* = {*q*1, *q*2,…, *qn*};

конечное множество входных символов Σ = {*a*1, *a*2,…, *am*};

* начальное состояние *qs*;
* множество допускающих состояний *F* = {*f*1, *f*1,…, *fk*};
* функция переходов δ(*q*1, *a*) = *q*2, сопоставляющая паре вида «состояние-символ» некоторое новое состояние.

 В теории автоматов используется определение конечного автомата в виде пятерки: A = (*Q*, Σ, δ, *qs*, *F*).

1. **Минимизация конечных автоматов**

Для применения в практике конечные автоматы стремятся упростить, минимизировать, чем он проще, тем меньше ресурсов потребуется для его материализации.

 Единственным параметром, отличающим более предпочтительный от

менее предпочтительного из двух автоматов, распознающих один и тот же язык, является количество состояний. А время функционирования автомата всегда пропорционально длине входной цепочки символов и не рассматривается при решении задач минимизации. В связи с этим автомат с меньшим числом состояний оказывается более оптимальным, поскольку требует меньшего объема памяти и устроен он, естественно, проще.

 Существует алгоритм, позволяющий получить оптимальный автомат, эквивалентный заданному, то есть получить автомат, выполняющий те же функции, но имеющий при этом минимально возможное количество состояний.

 В алгоритме используется понятие эквивалентности состояний.Два состояния называются эквивалентными, если последующее функционирование автомата в первом состоянии совпадает с последующим функционированием автомата во втором состоянии.

 Суть минимизации состоит в объединении эквивалентных состояний в единственное новое состояние.

 Алгоритм на псевдокоде выглядит следующим образом:

**удалить недостижимые состояния и связанные с ними правила;**

**создать таблицу всевозможных пар состояний вида (*p*, *q*);**

**ОТМЕТИТЬ те пары, где одно из состояний является допускающим, а другое –нет;**

**DO**

 **found = false;**

 **ЕСЛИ существует неотмеченная пара (*p*, *q*), такая, что для некоторого**

 **элемента входного алфавита *a* пара (δ(*p*, *a*), δ(*q*, *a*)) отмечена**

 **ОТМЕТИТЬ пару (*p*, *q*);**

 **found = true;**

**WHILE found // то есть пока изменения происходят**

**заменить каждое множество эквивалентных друг другу состояний на единственное новое; соответствующим образом изменить таблицу переходов**

 **5. Недетерминированные конечные автоматы**

Недетерминированный конечный автомат это устройство похожее на детерминированный конечный автомат, только в нем разрешены противоречащие друг другу правила переходов. Например, в недетерминированном автомате могут существовать правила 5,А → 3 и

5, А → 4 (рис. 2.) То есть, из состояния 5 под воздействием символа А правила автомата требуют перехода и в состояние 3 и в состояние 4.

*А*

 *А*

 Рис.2. Неоднозначность перехода недетерминированного автомата

 При описании функционирования автомата необходимо рассмотреть каждый вариант переходов.

 **6. ε-автомат**

Пусть имеется два недетерминированных автомата A и B. Первый автомат (А) имеет несколько допускающих состояний, а второй (B) – одно допускающее состояние. Задача состоит в том, чтобы соединить их в один автомат.

 Сложность решения задачи в том, что соединение, т. е. создание перехода от одного автомата к другому, должно быть помечено каким-либо символом входного алфавите, но автоматы уже созданы, и после того, как первый автомат обработал цепочки символов, должен запуститься второй без всяких промежуточных действий. Если бы первый автомат имел одно допускающее состояние, то можно было бы объединить это допускающее состояние с начальным состоянием второго, однако в данной ситуации это невозможно выполнить.

 Выход состоит в введении ε-переходов. Символом ε обозначается пустая строка и соответственно, ε-переход – это переход по пустой строке, то есть без считывания очередного символа входной последовательности символов.

Автомат содержащий ε-переход, называется ε-автоматом. Любойε-автомат относится к недетерминированным, поскольку ε-переход подразумевает возможность выбора – либо перейти в новое состояние без считывания символа входной последовательности, либо остаться в текущем состоянии.

 **7. Детерминизация недетерминированных автоматов**

 Пусть состояния исходного (недетерминированного) автомата помечены целыми числами от единицы до *n*. В алгоритме детерминизации в качестве основных будут использоваться рассматриваемые ниже процедуры.

 Одна из процедур, составляющая первый шаг детерминизации заключается в генерировании состояний нового (детерминированного) автомата. Всего таких состояний будет столько, сколько подмножеств у множества {1, 2, …, *n*}, не считая пустого множества, их будет 2*n* – 1. Это состояния {1}, {2}, …, {*n*}, {1, 2}, {1, 3},…, {1, *n*},…, {1, 2,…, *n*}. В этих множествах, один и тот же элемент не может присутствовать более одного раза, а так же, порядок следования элементов не имеет значения, т. е. {1, 2, 3} и {3, 2, 1} рассматриваются как одно и то же множество. При выполнении рассмотренной процедуры условились перечислять элементы множества только в порядке возрастания.

 Другая процедура состоит в построении, так называемого, множества достижимых по некоторому символу состояний.Допустим, что *s* – некоторое состояние, а *с* – некоторый символ входной цепочки. В детерминированном автомате любой паре (*s*, c) соответствует только одно новое состояние. В недетерминированном автомате из каждого состояния может исходить несколько переходов, помеченных одним и тем же символом, поэтому вместо единственного нового состояния мы получаем целое множество состояний *P*(*s*, *c*) достижимых из *s* по символу *c*.

 Третья процедура состоит в добавлении множеству  *P*(*s*, *c*) всех состояний, достижимых из его элементов посредством одного или более ε-переходов. Обозначается это множество через *P* ' (*s*, *c*).

 Рассмотрим применение указанных процедур на примере детерминизации автомата, представленного графом на рис.3.

*C*, *A*

*A*, ε

*C*

*A*

*С*

ε

ε

*A*

 Рис. 3. Исходный автомат для детерминизации

 Из стартового состояния по символу *A* можно попасть в состояния 3 и4, поэтому *P*(1, *A*) = {3, 4}. Из четвертого состояния по ε-переходам можно добраться до состояний 1 и 2; единственный ε-переход из состояния 3 ведет в состояние 5. Таким образом, *P* '(1, *A*) = {1, 2, 3, 4, 5}. Аналогично *P*(3, *C*) = {2}, *P* ' (3, *C*) = {1, 2}, а  *P*(1, *C*) = {2, 3},  *P* ' (1, *С*) = {1, 2, 3, 5}. Необходимо отметить, что элементы множества *P* ' строятся на основе элементов *P*, поэтому, *P* ' (3, *C*) не включает состояние 5, хотя оно достижимо из состояния 3 непосредственно по ε-переходу. Для автоматов не содержащих ε-переходов, элементы множеств *P* и  *P* ' совпадают.

 Стартовым состоянием нового автомата будет состояние, включающее все достижимые из *s* по ε-переходам состояния, помеченное множеством *s*, где *s –* стартовое состояние исходного автомата*.*  Состояние {1} было бы помечено как стартовое, поскольку стартовое состояние исходного автомата не имеет исходящих ε-переходов. Если бы в исходном автомате существовало правило 1,ε → 3, то стартовым состоянием результирующего автомата было бы состояние {1, 3, 5}.

 Допускающими состояниями создаваемого автомата будут состояния, запись которых включает в себя хотя бы одно допускающее состояние исходного автомата. В рассматриваемом примере автомат содержит лишь одно допускающее состояние – 5. Поэтому допускающими состояниями нового автомата будут состояния {5},{1, 5},{2, 5},{3, 5},{4, 5},{1, 2, 3},{1, 3, 5},{1, 4, 5},{2, 3, 5},{2, 4, 5},{3, 4, 5},{1, 2, 3, 5},{1, 2, 4, 5},{1, 3, 4, 5},{2, 3, 4, 5},{1, 2, 3, 4, 5}.

 Дальнейшие шаги алгоритма детерминизации на псевдокоде имеют вид:

для каждого состояния {*s*1, *s*2, …, *sn*} создаваемого автомата

 для каждого символа входного алфавита *с*

 *R* = {};

 для каждого элемента *s*i

 R = *R* ∪ *P* ' (*s*i, *c*);

 добавить в новый автомат правило {*s*1, *s*2, …, *sn*}, *c* → *R*;

 минимизировать полученный автомат;

 Это ядро алгоритма позволяет получить ответ на вопрос: в какие состояния можно попасть по символу *c* входной последовательности из исходного состояния автомата *s*1, *s*2, …, *sn*? Множество полученных состояний и будет определять единственное целевое состояние *R* создаваемого автомата.

 Для примера рассмотрим состояние {1, 3} и входной символ *A*. Так как

*P* ' (1, *A*) = {1, 2, 3, 4, 5}, а *P* ' (3, *A*) = {5}, то результирующее состояние R =

*P* ' (1, *A*) ∪ *P* ' (3, *A*) = {1, 2, 3, 4, 5}. Таким образом, получаем правило {1, 3}, *A* → {1, 2, 3, 4, 5}. Для состояния {2, 3, 5} и входным символом *C* получаем *R* = *P* ' (2, *C*) ∪ *P* ' (3, *C*) ∪ *P* ' (5, *C*) = {1, 2} поскольку *P* ' (2, *C*) = {}, *P* ' (3, *C*) = {1, 2}, а *P* ' (5, *C*) = {}.

 **8. Имитация функционирования автоматов**

 Для имитации функционирования недетерминированных автоматов

используются вычислительные средства. Идея имитации состоит в следующем. В любой ситуации, когда возникает недетерминированный переход, компьютер должен выбрать один из переходов и продолжить работу. Если после поступления всей входной последовательности автомат оказался в недопускающем состоянии, необходимо вернуться назад к месту «расщепления» переходов и пойти по другой ветви.

 При имитации недетерминированного автомата учитываются следующие особенности: 1) если программа на текущем шаге выбирает ε-переход, то считывания очередного входного символа не требуется; 2)считывание всех символов входной последовательности не означает завершения работы автомата; 3)если из состояния A в состояние B ведет ε-переход, и такой же переход ведет из B в A, то программа зацикливается; при этом на каждом шаге необходимо сохранять пару {состояние, номер текущего символа входной последовательности} и в случае повторения пары из сохраненных, переход следует исключить.

 **9. Программное средство имитации функционирования автоматов**

 Для имитации функционирования автоматов будем использовать инструмент JFLAP, выполненный на языке Java.

 Использовать JFLAP можно следующим образом.

 Двойной щелчок по JFLAP.jar запустит программу. В функциональном меню необходимо выбрать тип исследуемой в работе модели.

 Выбираем режим Finite Automaton.

 На экране появляется рабочая область, в которой можно сконструировать конечный автомат любого типа. Значок State Creator на панели инструментов позволяет создать новые состояния автомата, щелчок мышью в рабочей области приводит к появлению очередного состояния.

 В режиме Attribute Editor с помощью значка со стрелкой курсора щелкнув правой кнопкой по любому созданному состоянию, можно сделать его начальным (Initial) или допускающим (Final).

 Режим Transition Creator (значок со стрелкой) предназначен для создания переходов. Надо мышь навести на исходное состояние и нажатием и удержанием левой кнопки указать целевое состояние. После этого программа предлагает выбрать символ, соответствующий переходу. Если нажать Enter, то получится ε-переход, который в программе JFLAP назван λ-переходом, где в качестве метки пустой строки используется символ λ. Для создания перехода из состояния в него же, надо нажать левую кнопку мыши в одном месте кружочка-состояния, а отпустить в другом месте этого же кружочка. В программе переходы нельзя помечать несколькими символами. Для каждого символа должен быть свой переход. Программа сама оценит ситуацию и преобразует два перехода в один переход и пометит нужными символами.

 Готовый автомат можно протестировать, если задать какую-либо строку в качестве входной цепочки. Для этого используется пункт меню Input → Fast Run. Задав в качестве входа строку символов, получаем сообщение, что она допущена, либо не допущена, при этом указывается, какая именно последовательность переходов переводит автомат в допускающее состояние. Если существует несколько таких последовательностей, их можно вывести по очереди, нажимая кнопку Keep looking. Щелчок по кнопке I am done вернет управление в редактор JFLAP.

 Недетерминированный автомат можно детерминировать с использованием программы JFLAP. Для выполнения этого действия предназначен пункт меню Convert → Convert to DFA. Нажав кнопку Complete можно получить детерминированную версию автомата. Построенный автомат может выйти за пределы видимой области, сжать конструкцию в программе невозможно, ее можно только перетащить вручную.

 Щелчок Done переносит автомат в новое окно.

 В JFLAP имеются средства пошагового и ручного исполнения алгоритмов.

 Решение задачи минимизации так же можно выполнить с помощью программы JFLAP. Процедура минимизации запускается при выборе пункта меню Convert → Minimize DFA. Если минимизируемый автомат не полон, то есть не все переходы определены, программа автоматически исправит этот недостаток, направив недостающие переходы к специально созданному состоянию. JFLAP позволяет выполнить минимизацию по шагам, для этого необходимо щелкнуть по корню дерева в правом окне и нажать кнопку Complete Subtree. Различные терминальные узлы сгенерированного дерева будут соответствовать различным классам эквивалентных состояний.

 Кнопка Finish открывает в новом окне кружочки, соответствующие состояниям минимизированного автомата. Переходы пока не выводятся , поскольку JFLAP еще раз предлагает проверить знания и выполнить алгоритм нахождения правильных переходов вручную. Кнопкой Complete можно быстро закончить построение автомата, затем щелчком по кнопке Done перенести автомат в новое окно.

 **10.Преобразование регулярного выражения в конечный автомат**

Для превращения регулярного выражения в автомат необходимо рассмотреть представление в виде автоматов простейших атомарных конструкций регулярных выражений.

 1. Одиночный символ алфавита представляется в виде, показанном на рис. 4.

*F*

*a*

 Рис. 4. Автомат, распознающий одиночный символ

 2.Объединение двух регулярных выражений *a* и *b* (то есть (*a* ∪ *b*)) представлено на рис.5. Здесь использовано недетерминированное поведение.

*B*

*A*

ε

ε

 Рис. 5. Автомат, распознающий объединение двух

 регулярных выражений

3.Конкатенация двух регулярных выражений (*ab*) представлена на рис.6.

*A*

*B*

ε

ε

 Рис. 6. Автомат, распознающий конкатенацию

 двух регулярных выражений

3. Операция замыкание Клини реализуется автоматом представленным на рис.7.

ε

ε

A

ε

 Рис 7. Автомат, реализующий замыкание Клини

 В регулярных выражениях любая конкатенация или объединение обязательно заключаются в скобки. Поэтому в выражении можно найти самое внутреннее атомарное подвыражение и представить его в виде автомата. Затем перейти к внешнему подвыражению и так далее продолжить конвертирование регулярного выражения в автомат.

 Пример конвертирования выражения ((*a* ∪ *b*)\* *с*) представлен на рис. 8.

*a*

*b*

ε

 *a*

 *b*

ε

ε

ε

ε

*a*

*b*

*a*

ε

 ε

 ε

ε

 ε

 *b*

 *c*

 Рис. 8. Автомат для регулярного выражения ((*a* ∪ *b*)\* *с*)

 **11. Преобразование конечного автомата в регулярное выражение**

 Рассмотрим свойства ε-строк, которые позволяют «сжимать» строки, содержащие ε-подстроки.

1. конкатенация любого количества пустых строк представляет собой пустую строку: εε…ε = ε;
2. конкатенация пустой строки ε с непустой строки *a* есть строка *a*:

ε*a* = *a*ε = *a*.

 Так, например строка может быть следующим образом «сжата»:

 *aa*ε*b*εεεεε*c*εε*dd*ε = *aabcdd* .

 Для того, чтобы автомат был пригоденн к переводу в регулярное выражение, от должен обладать следующими свойствами:

1. ни один переход недолжен вести в стартовое состояние *S*. В противном случае следует ввести новое стартовое состояние *S* ' и добавить правило перехода *S* ', ε → *S*;
2. автомат должен содержать лишь одно допускающее состояние. Запрещены переходы ведущие из допускающего состояния. В противном случае допускающее состояние вычеркивается из списка допускающих и вводится новое допускающее состояние *F* ' , а затем добавляется переход *F*, ε → *F* '.

Суть алгоритма преобразования состоит последовательной замене

состояний соответствующими им регулярными выражениями. Процесс замены продемонстрирован на рис. 9.

*b*

*ab\*c*

*a*

*c*

 Рис.9. Замена состояния регулярным выражением

 Изменяемый в процессе построения регулярного выражения автомат называется диаграммой переходов (или графом), поскольку переходы в нем уже могут осуществляться не по одному символу, а по целым регулярным выражениям.

 Алгоритм преобразования на псевдокоде имеет следующий вид.

**ДЛЯ ВСЕХ ребер графа автомата**

 **ЕСЛИ метка ребра равна *a*1, *a*2,…, *an***

 **заменить метку на (*a*1 ∪ *a*2 ∪… ∪ *an*)**

**ДЛЯ КАЖДОГО узла диаграммы *K* (кроме стартового и допускающего)**

 **ДЛЯ КАЖДОГО узла *A* (*A* ≠ *K*)**

 **ДЛЯ каждого узла *B* (*B* ≠ *K*)**

 **ЕСЛИ существуют ребра (*A*, *K*) и ( *K, B*)**

 ***Mak* = метка ребра (*A*, *K*);**

 ***Mkb* = метка ребра ( *K, B*);**

 **ЕСЛИ существует ребро (*K*, *K*) (*Mkk* = метка ребра (*K*, *K*))**

 ***M* = *Mak Mkk* \* *Mkb*;**

#  **ИНАЧЕ**

 ***M* = *Mak Mkb*;**

 **ЕСЛИ существует ребро (*A*, *B*) (*Mab* = метка ребра (*A*, *B*))**

 **метка ребра (*A*, *B*) = *Mab* ∪ *M***

**ИНАЧЕ**

 **метка ребра (*A*, *B*) = *M*;**

 **КОНЕЦ ЦИКЛА**

 **КОНЕЦ ЦИКЛА**

 **удалить узел *K* и связанные с ним ребра;**

 **КОНЕЦ ЦИКЛА**

 В итоге выполнения алгоритма будут удалены все узлы кроме стартового и допускающего, а метка оставшегося ребра будет содержать ответ. Результирующее регулярное выражение следует упростить, исключив пустые символы.

  **12. Выполнение задания**

  **№1:** 1) построить автомат согласно формального описания, представленного в индивидуальном задании; 2) задать строку входных символов и протестировать автомат; 2) детерминизировать недетерминированный автомат; 3) минимизировать детерминированный автомат; 4) преобразовать конечный автомат в регулярное выражение; 5) преобразовать регулярное выражение в конечный автомат.

  **№2:** выполнить пункты задания №1 с применением программы JFLAP.

 Индивидуальные задания

1. *A* = (*Q*, Σ, δ, *qs*, *F*), где *Q* = {1,2,3,4}; Σ = {*a*,*b*}; δ = {‹1,*a*,3›,‹1,*a*,2›, ‹2,*a*,2›, ‹2,*b*,2›, ‹3,*a*,2›, ‹3,*b*,4›, ‹4,*a*,3›, ‹4,*b*,1›}; *qs* = {1}; *F* = {1,3}.

 (скачать свободно распространяемую программу можно на сайте www. cs. duke. Edu/ ~rodger/ tools/ jflap/.

и требует установки Java Runtime Environmtnt (JRE) с вебсайта [www.java.sun.com](http://www.java.sun.com/).