Контрольная работа

Методические указания по выполнению.

Контрольная работа должна выполняться после изучения всего теоретического материала. Решения задач следует сопровождать пояснениями. Контрольная работа выполняется в электронном виде и высылается со страниц сайта ДО. В случае наличия ошибок контрольная возвращается студенту на доработку.

**Внимание!!!**

***Работу над ошибками необходимо выполнять в том же файле, сохраняя замечания преподавателя!!!***

Задачи №№1–4 относятся к теме регулярных языков (*материал глав 1 и 2*). Для описанного в первой задаче регулярного языка требуется построить различные конструкции, задающие этот язык – регулярное выражение, регулярную и КС-грамматики, детерминированный конечный автомат.

В первой задаче необходимо построить регулярное выражение, задающее словесно описанный язык. (*Материал разделов 1.1, 2.1*).

Например: Пусть язык задан словесным описанием «Множество цепочек из {0,1,a,b}\*, заканчивающихся цепочкой ’01a’ и содержащих чётное количество символов ‘a’». Требуется построить для этого языка регулярное выражение. Для начала определим минимально возможную цепочку этого языка. Самая короткая возможная цепочка ‘a01a’ – она заканчивается требуемой цепочкой ’01a’ и содержит 2 символа ‘a’. Это основа нашего регулярного выражения. Удлинить эту цепочку можно, поместив перед ней или перед цепочкой ’01a’ в ней любое количество любых символов алфавита, за исключением символа ‘a’ – для него нужно подсчитывать количество, поэтому его надо рассматривать особо. Заданную цепочку ’01a’ разрывать нельзя. Для генерации любого количества символов используется итерация, а выбор одного из нескольких символов записывается в виде суммы. Итак, получим (0+1+b)\*a(0+1+b)\*01a. Но по построенному таким образом выражению нельзя получить много символов ‘a’. Для увеличения количества ‘a’ можно добавить перед построенным выражением (aa)\* для сохранения чётности количества символов ‘a’. Но при добавлении такой конструкции нельзя будет между символами ‘a’ поместить другие символы алфавита. Для того чтобы сделать это возможным, заменим добавляемую конструкцию на ((0+1+b)\*a(0+1+b)\*a(0+1+b)\*)\*, сохранив чётность ‘a’ и добавив любое количество других символов. В общем случае, полученную конструкцию следовало бы поместить и после первого символа ‘a’ в ранее построенном выражении (вместо выделенной цветом скобки), но в рассматриваемом языке это ничего не изменит, только усложнит выражение. Поэтому итоговое регулярное выражение будет иметь вид: ((0+1+b)\*a(0+1+b)\*a(0+1+b)\*)\*(0+1+b)\*a(0+1+b)\*01a. Можно несколько упростить построенное выражение, убрав одну из скобок (0+1+b)\* – там, где она повторяется. Тогда получим окончательный ответ: **((0+1+b)\*a(0+1+b)\*a)\*(0+1+b)\*a(0+1+b)\*01a**.

Во второй задаче для того же самого языка требуется построить регулярную грамматику. Поскольку в задании её тип не указан явно, можно строить как праволинейную, так и леволинейную. (*Материалы разделов 1.2 и 1.3*). Будем для определённости строить праволинейную грамматику, т.е. в правой части правил вывода нетерминальные символы размещать только справа от терминальных цепочек. Для повторения символа алфавита любое количество раз используется явная рекурсия на нетерминальном символе (вида A→0A – тогда ‘0’ может повториться хоть сколько раз), для подсчёта чётности применяется чередование двух нетерминалов (A→0B, B→0A). Начнём с минимальной цепочки ‘a01a’. Поскольку она начинается с символа ‘a’, количество которых необходимо подсчитывать, после его генерации нужно изменить нетерминал. Далее идёт неразрываемая обязательная цепочка ‘01a’, следовательно, её нужно получать всю целиком. Это позволит сделать грамматика с правилами S→aA, A→01a. Для того чтобы получить любое количество символов алфавита перед ‘a01a’, добавим рекурсивные правила для целевого символа S: S→0S|1S|bS. Для получения таких же символов после первого ‘a’ в построенной минимальной цепочке добавим такие же рекурсивные правила для нетерминала A: A→0A|1A|bA. Правила грамматики приняли вид:

S→aA|0S|1S|bS, A→01a|0A|1A|bA.

Осталось дополнить правила генерацией чётного количества символов ‘a’. Поскольку нетерминал A означает, что получен один символ ‘a’ в итоговой цепочке, а S – что нет ещё ни одного символа ‘a’, то можно использовать S для получения любого чётного их количества, а А – любого нечётного, просто чередуя S и A: A→aS, S→aA, причём второе из этих правил уже есть. Грамматика построена, выпишем её полностью. **G({0,1,a,b},{S,A},P,S)**, где правила P:

**S→0S|1S|bS|aA,**

**A→0A|1A|bA|aS|01a**.

В третьей задаче требуется построить КС-грамматику для этого же языка (*разделы 1.2, 1.3, 1.4*). Её правила удобно строить по регулярному выражению. Для этого каждую большую скобку обозначаем своим нетерминалом. Если на скобке стоит звёздочка (итерация), значит, на этом нетерминале будет явная рекурсия и пустое правило. Регулярное выражение имело вид ((0+1+b)\*a(0+1+b)\*a)\*(0+1+b)\*a(0+1+b)\*01a. Обозначим A=((0+1+b)\*a(0+1+b)\*a)\*, B=(0+1+b)\*. Тогда первое правило будет иметь вид: S→ABaB01a. В правиле для A будут присутствовать B: A→BaBa, надо добавить рекурсию и пустое правило: A→BaBaA|λ. Нетерминал B рекурсивно порождает любые символы кроме ‘a’: B→0B|1B|bB|λ. Итак, грамматика построена, выпишем её полностью.

**G({0,1,a,b},{S,A,B},P,S)**, где P:

**S→ABaB01a,**

**A→BaBaA|λ,**

**B→0B|1B|bB|λ.**

Эта грамматика более наглядна, чем регулярная.

Ещё в этом же задании требовалось сгенерировать цепочки языка по полученной грамматике. Покажем на примере одной цепочки, подчёркивая каждый раз нетерминал, который будем раскрывать, и используя правосторонний вывод (каждый раз заменяем крайний правый нетерминал в цепочке вывода).

S ⇒ ABaB01a ⇒ (правило B→λ) ABa01a ⇒ (правило B→0B) A0Ba01a ⇒ (правило B→λ)  A0a01a ⇒ (правило A→BaBaA) BaBaA0a01a ⇒ (правило A→λ) BaBa0a01a ⇒ (правило B→λ)  Baa0a01a ⇒ (правило B→λ) **aa0a01a**.

Получили цепочку, соответствующую требованиям языка.

В четвёртой задаче для этого же языка требуется построить распознаватель – детерминированный конечный автомат (ДКА) (*разделы 1.5 и 2.2*). При построении ДКА полезно следовать трём правилам:

1) Если распознаваемый язык допускает пустую цепочку, то начальное состояние автомата является также и конечным.

2) Минимальная непустая цепочка языка должна переводить автомат из начального состояния в конечное.

3) Если все цепочки языка должны иметь длину заданной кратности, то в функции переходов количество состояний в цикле равно этой кратности (и не может быть меньшим!!!). Тогда для цепочек произвольной длины на вершинах графа функции переходов допустимы петли, для чётной длины циклы включают по два состояния, и т.д.

Для наглядности построение функции переходов ДКА будем выполнять сначала в виде графа. Построение начинается так же – с минимальной цепочки заданного языка – ‘a01a’, которая является основой будущего автомата. Каждый символ цепочки читается за один переход из состояния в состояние. При этом после прочтения минимальной цепочки ДКА должен оказаться в конечном состоянии, которое обозначаем двойным кружком.

q0

q1

q4

1

a

q2

q3

0

a

Охарактеризуем все построенные состояния с точки зрения основных ограничений заданного языка – количества прочитанных символов ‘a’ и наличия требуемой цепочки. Состояние q0 будет характеризоваться «чётное количество символов ‘a’, заданной цепочки ещё нет». Состояние q1 – «нечётное количество символов ‘a’, заданной цепочки ещё нет». Состояние q2 – «нечётное количество символов ‘a’, прочитан первый символ ‘0’ заданной цепочки». Состояние q3 – «нечётное количество символов ‘a’, прочитано первые два символа ‘01’ заданной цепочки». Состояние q4 – «чётное количество символов ‘a’, прочитана вся заданная цепочка».

Теперь для каждого из построенных состояний рассматриваем возможные переходы по всем остальным символам алфавита, учитывая, что любой посторонний символ между символами цепочки ‘01a’ вызовет возвращение в предыдущие состояния. Поскольку автомат детерминированный, из любого состояния по каждому из символов алфавита возможно делать только по одному переходу. Начинаем с q0. В нём можно прочитать хоть сколько любых символов, кроме ‘a’; поэтому делаем цикл – рисуем петлю для ‘0,1,b’. В состоянии q1 – точно так же, но переход по ‘0’ из него уже задан. Поэтому в цикле можно использовать только ‘1,b’. Для того чтобы получить любое количество нулей, зацикливаемся по нулю в q2. Получаем:

q0

q1

q4

1

a

0,1,b

q2

q3

0

a

1,b

0

В состоянии q1 осталось рассмотреть символ ‘a’. Поскольку эти символы нужно подсчитывать, после прочтения ‘a’ (это будет уже второй прочитанный символ ‘a’) необходимо вернуться в состояние их чётного количества – q0. Из состояния q2 осталось задать переходы по ‘b’ и по ‘a’. Прочтение ‘b’ нарушает цепочку ‘01a’, но не меняет количества ‘a’, значит, нужно вернуться в q1, где было столько же ‘a’ и ещё не прочитана цепочка. Прочтение ‘a’ нарушает цепочку и меняет количество ‘a’ – значит, нужно вернуться в q0.

q0

q1

q4

1

a

0,1,b

q2

q3

0

a

a

1,b

0

a

b

Рассмотрим состояние q3. В нём ДКА окажется после прочтения первых двух символов ‘01’ нашей цепочки ‘01a’, значит, любой из символов кроме ‘a’ её нарушит. Но если это будет символ ‘0’ – т.е. прочитано ‘a010’, то тем самым уже есть первый символ нашей цепочки, а это состояние q2. Значит, по ‘0’ надо вернуться в q2. Символы ‘1’ и ‘b’ равноправны, они требуют заново читать цепочку и не меняют количества ‘a’, поэтому по ним одинаковый переход – в q1.

q0

q1

q4

1

a

0,1,b

q2

q3

0

a

a

1,b

0

a

b

0

1,b

Рассуждая совершенно аналогичным образом, построим все переходы из конечного состояния q4. Если нужно прочесть ‘a’, то получится цепочка ‘a01aa’ – надо перейти в состояние другой чётности, чем q4, и при этом снова нужна вся цепочка. Это q1. Если нужно прочесть ‘0’, то получится цепочка ‘a01a0’ – есть начало цепочки и, казалось бы, это состояние q2, но количество ‘a’ не изменилось, а в q2 их было нечётное количество. Значит, придётся вернуться в состояние той же чётности – q0. По остальным символам – так же. В итоге получим:

q0

q1

q4

1

a

0,1,b

q2

q3

0

a

a

1,b

0

a

b

0

1,b

a

0,1,b

Выпишем полностью построенный ДКА: **M({q0,q1,q2,q3,q4},{0,1,a,b},δ,q0,{q4})**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***вход*** | | | | Для наглядности закрасим ячейки таблицы, соответствующие переходам минимальной допустимой цепочки.  Рассмотрим разбор построенной в задаче №3 цепочки ‘aa0a01a’: |
| ***состояние*** | 0 | 1 | a | b |
| q0 | {q0} | {q0} | {q1} | {q0} |
| q1 | {q2} | {q1} | {q0} | {q1} |
| q2 | {q2} | {q3} | {q0} | {q1} |
| q3 | {q2} | {q1} | {q4} | {q1} |
| q4 | {q0} | {q0} | {q1} | {q0} |

Осталось согласно заданию представить функцию переходов ещё и в виде таблицы, а также проверить по автомату построенные в предыдущей задаче цепочки.

(q0,aa0a01a) ├─ (q1,a0a01a) ├─ (q0,0a01a) ├─ (q0,a01a) ├─ (q1,01a) ├─ (q2,1a) ├─ (q3,a) ├─ (q4,λ) Поскольку q4∈F, это означает, что ‘aa0a01a’∈L(M).

Замечание.

При построении ДКА (например, если в описании языка задана начальная цепочка) может возникнуть ситуация, когда в некоторых состояниях возможно определить переходы не для всех символов алфавита. В таком случае допустимо либо оставить автомат неполностью определённым, либо ввести дополнительное состояние «ошибка», в которое направить все невозможные переходы.

Например, пусть язык задан алфавитом {0,1,2} и все цепочки должны начинаться с пары ‘01’ и содержать нечётное количество единиц. Минимальная допустимая цепочка ‘01’.

q0

q1

q2

0

1

После этой цепочки может быть прочитано любое количество ‘0’ и ‘2’ или чётное количество единиц. Значит, можно использовать цикл на состоянии q2 для ‘0’ и ‘2’ и чередование двух состояний для единиц. НО!!! Использовать для этого имеющееся состояние q1 нельзя, т.к. из него невозможно прочесть ‘0’ или ‘2’. Поэтому придётся вводить новое состояние, которое нужно только для подсчёта единиц, а остальные символы в нём могут быть прочитаны в любом количестве.

q0

q1

q2

1

0

q3

1

0,2

0,2

1

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***вход*** | | |
| ***состояние*** | 0 | 1 | 2 |
| q0 | {q1} |  |  |
| q1 |  | {q2} |  |
| q2 | {q2} | {q3} | {q2} |
| q3 | {q3} | {q2} | {q3} |

Функцию переходов можно оставить в таком виде, когда переходы из состояний q0 и q1 по остальным символам не определены. Тогда в таблице останутся пустые ячейки:

Другой способ – можно доопределить автомат ещё одним состоянием «ошибки» qerr. Это состояние, в которое можно зайти по любому «постороннему» символу, но из которого уже нельзя выйти. Оказавшись в этом состоянии, автомат уже не сможет ни после каких символов попасть в конечное состояние, следовательно, цепочка не будет принята. Прочтение, например, цепочки ‘0001’ вызовет именно такой переход и автомат останется в состоянии qerr.

(q0,0001) ├─ (q1,001) ├─ (qerr,01) ├─ (qerr,1) ├─ (qerr,λ). Цепочка дочитана, но qerr∉F, следовательно ‘0001’∉L(M).

q0

q1

q2

1

0

q3

1

0,2

0,2

1

qerr

1,2

0,2

0,1,2

Задачи №№ 1–4 можно решать в любом удобном порядке. Например, иногда более удобным оказывается сначала построить ДКА, а уже затем по нему достаточно просто строится регулярная грамматика. Так, если в автомате есть переход то в грамматике ему соответствует правило A→0B.

A

B

0

Задачи №№5,6 относятся к теме КС-языков (*материал глав 1 и 3*). Для описанного в пятой задаче контекстно-свободного языка требуется построить различные конструкции, задающие этот язык.

В пятой задаче требуется построить КС-грамматику по заданному описанию языка и затем сгенерировать цепочки по построенной грамматике, выписав процесс генерации (*разделы 1.2, 1.3, 1.4*).

Например: Пусть язык задан следующим описанием: L={w⏐w∈{a, b}\* и количество символов ’а’ и ’b’ одинаково и чётно}. К этому языку будут относиться цепочки вида ‘abba’, ‘aabb’, ‘abbabaab’, ‘bbbaaaab’… Требуется построить КС-грамматику, задающую этот язык. Для гарантии равенства количества символов ’а’ и ’b’ проще всего получать их в одном правиле вывода, например: S→aSb. Поскольку количество каждого из символов должно быть чётным, нужно чередовать нетерминалы: S→aAb, A→aSb. Символы ’а’ и ’b’ могут идти в любом порядке, поэтому необходимо предусмотреть правила для порождения их как «изнутри» цепочки – как в выписанных правилах (будут строиться цепочки вида ‘aabb’), так и «снаружи», и в любом порядке: S→abA|baA. В итоге получаем правила: S→aAb|abA|bAa|baA; A→aSb|abS|bSa|baS. Единственный недостаток построенных правил – не предусмотрено выхода из рекурсии. Из задания следует, что язык допускает пустую цепочку. Значит, разрешается пустое правило для целевого символа. Окончательный вид грамматики: **G({a,b},{S,A},P,S)**, где P:

**S → aAb|abA|bAa|baA|λ,**

**A→ aSb|abS|bSa|baS.**

Осталось построить цепочки языка по получившейся грамматике. Для примера рассмотрим получение одной цепочки. Поскольку нетерминал на каждом шаге вывода только один, для наглядности будем подчёркивать полученную из него цепочку:

S ⇒ aAb ⇒ aabSb ⇒ aabbaAb ⇒ aabbabaSb ⇒ **aabbabab**.

S ⇒ abA ⇒ abbSa ⇒ **abba**.

В шестой задаче нужно построить распознаватель с магазинной памятью для того же языка, что был задан в пятой задаче, и проверить допустимость этим распознавателем построенных в задаче №5 цепочек (*разделы 1.5, 3.1*).

Язык рассматриваем тот же: L={w⏐w∈{a, b}\* и количество символов ’а’ и ’b’ одинаково и чётно}. При построении функции переходов ДМПА потребуется учитывать два фактора – равенство количества символов ‘a’ и ‘b’ и их чётность. Стандартный приём для отслеживания равенства количества символов – записывать символы одного вида в стек и затем удалять их оттуда по одному при появлении на входе альтернативного символа. Когда стек пустой, в него заносится первый встреченный символ. Для контроля чётности достаточно чередовать состояния. Исходя из этого, определим два состояния q0, q1 – состояние q0 будет означать, что прочитано чётное количество пар (a,b), причём неважно, в каком порядке; состояние q1 – что прочитано нечётное количество таких пар (a,b). В любом из этих состояний будем производить запись в стек без смены состояния, если символ на входе совпадает с символом стека, и удаление из стека со сменой состояния – если символы на входе и в стеке различны. Поскольку язык допускает пустую цепочку, конечным состоянием можно сделать состояние q0. При таком подходе для решения задачи с альтернативным вариантом языка – цепочки с нечётным одинаковым количеством ‘a’ и ‘b’ – достаточно изменить конечное состояние ДМПА с сохранением функции переходов. Итак, функция переходов δ будет иметь вид:

|  |  |  |  |
| --- | --- | --- | --- |
| δ(q0,a,Z)={(q0,aZ)} | δ(q1,a,Z)={(q1,aZ)} | δ(q0,a,b)={(q1, λ)} | В этих правилах символы удаляются из стека со сменой состояния. |
| δ(q0,b,Z)={(q0,bZ)} | δ(q1,b,Z)={(q1,bZ)} | δ(q0,b,a)={(q1, λ)} |
| δ(q0,a,a)={(q0,aa)} | δ(q1,a,a)={(q1,aa)} | δ(q1,a,b)={(q0, λ)} |
| δ(q0,b,b)={(q0,bb)} | δ(q1,b,b)={(q1,bb)} | δ(q1,b,a)={(q0, λ)} |
| В этих правилах входные символы переписываются в стек без изменения состояния. | | δ(q0,λ,Z)={(q0, λ)} | Конец работы. |
|  |  |

Осталось выписать полностью ДМПА P и проверить допустимость полученной в предыдущей задаче цепочки, для примера возьмём ‘aabbabab’.

P({q0,q1},{a,b},{Z,0},δ,q0,Z,{q0});

(q0,aabbabab,Z) ├─ (q0,abbabab,aZ) ├─ (q0,bbabab,aaZ) ├─ (q1,babab,aZ) ├─ (q0,abab,Z) ├─ (q0,bab,aZ) ├─ (q1,ab,Z) ├─ (q1,b,aZ) ├─ (q0,λ,Z) ├─ (q0,λ,λ).

Поскольку q0∈F, это означает, что ‘aabbabab’∈L(P).

В седьмой задаче требуется для заданного перевода построить синтаксически управляемую схему, которая будет этот перевод выполнять (*раздел 4.2.1*).

Например: Пусть надо выполнить перевод, который определён следующим образом: {(x,y)| x=1k02n12n, y=a3nbnck, n≥0, k>0}. Это означает, что в цепочках исходного языка стоит сначала произвольное количество единиц (ненулевое), потом – чётное количество нулей (2n), причём их может и не быть (допустимо n=0), и потом столько же единиц. Эти символы должны быть размещены в строго описанном порядке. Цепочка такого вида должна быть переведена в цепочку, где сначала подряд идёт 3n символов ‘a’, затем n символов ‘b’ и после этого символы ‘c’ в таком же количестве, что было единиц в начале цепочек исходного языка. Например, цепочка ‘100001111’ (n=2, k=1) должна быть переведена в ‘aaaaaabbc’, цепочка ‘10011’ – в ‘aaabc’, ‘1110011’ – в ‘aaabccc’, ‘1’ (n=0, k=1) – в ‘c’.

Построение СУ-схемы полезно начать с построения правил входной грамматики. Входная цепочка логически может быть разделена на две составных части – первая часть с произвольным количеством единиц, и вторая, где нужно отслеживать равное (и чётное) количество нулей и единиц. Имеет смысл для каждой из этих частей завести свой нетерминал, например S→AB. Тогда для первой части цепочки достаточно явной рекурсии вида A→1A c правилом A→1 для выхода из рекурсии, а вторая часть цепочки должна “разрастаться” из середины, причём в каждом правиле должно генерироваться сразу по два символа каждого вида: B→00B11. Выход из рекурсии произойдёт по пустому правилу B→λ, т.к. по условию задачи этих символов может и не быть. Окончательно правила входной грамматики будут иметь вид: S→AB, A→1A|1, B→00B11|λ. Рассуждая аналогичным образом, рассмотрим, какие элементы перевода должны соответствовать правилам вывода. Результирующая цепочка так же распадается на две части, где последняя часть – символы ‘c’ – должны соответствовать по количеству первой части исходной цепочки. Поэтому нетерминалы в элементе перевода должны стоять в обратном порядке. Что касается второй части исходной цепочки, то из двух её нулей (или единиц,) должно получаться три символа ‘a’ и один символ ‘b’. На основании сказанного запишем: S→AB,BA; A→1A,cA; A→1,c; B→00B11,aaaBb; B→λ,λ. Итоговая схема будет иметь вид:

T = ({S,A,B}, {0,1}, {a,b,c}, R, S), правила перевода R:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (1) S→AB,BA; | (2) A→1A,cA; | (3) A→1,c; | (4) B→00B11,aaaBb; | (5) B→λ,λ. |

Рассмотрим перевод одной из цепочек, например, ‘100001111’.

(S,S) ⇒(1) (AB,BA) ⇒(3) (1B,Bc) ⇒(4) (100B11,aaaBbc) ⇒(4) (10000B1111,aaaaaaBbbc) ⇒(5) (**100001111,aaaaaabbc**). Получили пару, где первый элемент – входная цепочка, второй элемент – её перевод.

В восьмой задаче нужно построить преобразователь с магазинной памятью (*раздел 4.2.2*) для выполнения того же перевода, который задан в предыдущей задаче. Напомним, что преобразователь – это распознаватель с выходом.

Сначала распознаватель должен прочитать любое – ненулевое – количество единиц. В принципе распознавателю можно при этом не использовать стек, а только убедиться в том, что была хотя бы одна единица (можно было бы для этого сменить состояние). Но распознаватель является ещё и преобразователем, а при генерации выходной цепочки в её конце нужно будет выдать столько же символов ‘c’. Следовательно, считываемые единицы нужно запоминать – записывать в стек, не производя при этом никакого выхода. Можно делать это, оставаясь в начальном состоянии, т.к. признаком наличия хотя бы одной единицы будет единица, записанная в стек. При появлении на входе первого нуля необходимо сменить состояние, т.к. меняется смысл действий. Теперь нужно будет заносить нули в стек – для проверки, что их количество совпадёт с последующими единицами; для проверки чётности этого числа удобно при записи нулей в стек чередовать состояния, например, q1 и q2. При этом на каждые два прочитанных нуля следует выдавать в выходную цепочку три символа ‘a’. Когда во входной цепочке снова появятся единицы, распознаватель должен находиться в состоянии q2, т.к. это состояние, в которое он попадёт после прочтения чётного количества нулей. Поскольку смысл действий снова меняется, опять нужна смена состояния. Далее можно было бы просто продолжать читать единицы с входной ленты, удаляя из стека нули, и если единицы и нули закончатся одновременно, то цепочка соответствует требуемой. Но помимо проверки цепочки, нужно выполнять перевод, а в выходной цепочке двум нулям стека (или двум единицам на входе, что равносильно) должен соответствовать один символ ‘b’. Значит, придётся снова чередовать состояния, например, q3 и q4, чтобы производить выход только в одном из двух чередуемых состояний. После окончания цепочки нужно удалить из стека единицы, выдав по символу ‘c’ на каждую. Если в исходной цепочке было n=0, то по окончании цепочки надо перейти из q0 к состоянию q5 выдачи на выход символов ‘c’. Итак, преобразователь:

P =({q0,q1,q2,q3,q4,q5},{0,1},{0,1,z0},{a,b,с},δ,q0,z0,{q5}),

|  |  |
| --- | --- |
| 1) δ(q0,1,z0)={(q0,1z0,λ)} | Считывание первых единиц входной цепочки и запись их в стек. Выхода нет. |
| 2) δ(q0,1,1)={(q0,11,λ)} |
| 3) δ(q0,0,1)={(q1,01,a)} | Считывание нулей входной цепочки, запись их в стек с чередованием состояний, выдача на выходе трёх ‘a’ на каждые два нуля входа. |
| 4) δ(q1,0,0)={(q2,00,aa)} |
| 5) δ(q2,0,0)={(q1,00,a)} |
| 6) δ(q2,1,0)={(q3,λ,b)} | Считывание единиц входной цепочки, удаление из стека нулей с чередованием состояний, выдача на выходе одного ‘b’ на каждые два нуля стека. |
| 7) δ(q3,1,0)={(q4,λ,λ)} |
| 8) δ(q4,1,0)={(q3,λ,b)} |
| 9) δ(q4,λ,1)={(q5,λ,c)} | Удаление из стека единиц, выдача на выход ‘c’ на каждую единицу стека. Входа нет, работа со стеком. |
| 10) δ(q5,λ,1)={(q5,λ,c)} |
| 11) δ(q5,λ,z0)={(q5,λ,λ)} | Окончание работы, опустошение стека. |
| 12) δ(q0,λ,1)={(q5,λ,c)} | Случай n=0. |

Проверим ту же цепочку, перевод которой рассматривали в предыдущей задаче.

Переведём цепочку ‘100001111’. Для удобства будем на каждом такте работы подписывать номера использованных правил.

(q0,100001111,z0,λ) ├─ (1) (q0,00001111,1z0,λ)├─ (3) (q1,0001111,01z0,a) ├─ (4) (q2,001111,001z0,aaa) ├─ (5) (q1,01111,0001z0,aaaa) ├─ (4) (q2,1111,00001z0,aaaaaa) ├─ (6) (q3,111,0001z0,aaaaaab) ├─ (7) (q4,11,001z0,aaaaaab) ├─ (8) (q3,1,01z0,aaaaaabb) ├─ (7) (q4,λ,1z0,aaaaaabb) ├─ (9) (q5,λ,z0,aaaaaabbс) ├─ (11) (q5,λ,λ,**aaaaaabbс**).

Преобразователь пришёл в конечное состояние, цепочка дочитана, стек пуст. Следовательно, входная цепочка распознана и построен её перевод – цепочка, полученная на выходе.

Итак, по каждой из задач контрольной работы приведено примерное решение. При выполнении контрольной необходимо делать аналогичные пояснения к каждому решению, хотя можно и не столь подробные как в методических указаниях. Правила грамматик и правила функции переходов автомата (преобразователя) с магазинной памятью рекомендуется нумеровать.

## Вариант 9

**№1** Пусть регулярный язык задан своим описанием:

Множество всех цепочек из {1,a,b}\*, содержащих подцепочку ’1b’ и нечётное количество единиц. Например, ‘1b’, ‘11b1’, ‘1a1b1’ и т.п.

Построить регулярное выражение, задающее этот язык.

**№2** Построить регулярную грамматику, задающую язык из задачи №1.

**№3** Построить КС-грамматику, задающую язык из задачи №1. Сгенерировать две цепочки языка по построенной грамматике. Процесс генерации цепочек языка записать в виде цепочки вывода, указывая номера применённых правил (или сами правила, как показано в примере). Использовать левосторонний или правосторонний вывод.

**№4** Построить детерминированный конечный автомат (ДКА), распознающий язык из задачи №1. Функцию переходов ДКА представить в двух видах: таблицей и графом переходов. Проверить с помощью этого ДКА допустимость цепочек языка, полученных в задаче №3. Процесс проверки выписать в виде последовательности конфигураций построенного ДКА.

**№5** Пусть КС-язык задан своим описанием:

L={a 2nb kc k ⏐ k > 0, n ≥ 0}. Например, ‘bc’, ‘aabc’, ‘aabbcc’, ‘aaaabc’.

Построить КС-грамматику, задающую этот язык. Допустимо использовать пустые правила. Сгенерировать две цепочки языка по построенной грамматике. Процесс генерации цепочек языка записать в виде цепочки вывода, указывая номера правил.

**№6** Построить детерминированный автомат с магазинной памятью, распознающий язык из задачи №5 и работающий с опустошением стека. Проверить с помощью этого ДМПА допустимость цепочек языка, полученных в задаче №5. Процесс проверки выписать в виде последовательности конфигураций построенного ДМПА, указывая номера правил.

**№7** Пусть требуется выполнить перевод τ цепочек с одного КС-языка на другой:  
τ = {(x,y) | x = ak+1bk1n, y=02k1kan | k≥0, n>0}. Например: (a1,a), (aab1,001a).

Построить T – схему синтаксически управляемого перевода для выполнения этого τ(T). Взять две цепочки исходного языка и выполнить их перевод, процесс перевода выписать в виде выводимых пар цепочек, указывая номера правил.

**№8** Построить преобразователь с магазинной памятью P для выполнения перевода τ(P) из задачи №7. Взять две цепочки исходного языка и выполнить их перевод, процесс перевода выписать в виде последовательной смены конфигураций построенного преобразователя, указывая номера правил.