Последние цифры пароля 21.

Сюда входит 5 лабораторных работ и курсовая.

Курсовая работа

Методические указания по выполнению

Курсовая работа должна выполняться после изучения всего теоретического материала и выполнения лабораторных работ. Курсовая работа состоит в написании программы в соответствии с заданием. В случае наличия ошибок в программе она возвращается на доработку.

**При выполнении работы над ошибками необходимо сохранять замечания преподавателя, а изменения в отчёт вносить другим цветом.**

Выполнение задания включает разработку программного средства, тестирование его на наборе данных и написание отчёта по работе.

Отчёт должен содержать:

1. титульный лист;
2. номер варианта и текст задания;
3. описание алгоритма решения задачи с иллюстрацией его на своём примере;
4. описание основных блоков программы;
5. текст программы;
6. результаты тестирования программы;
7. распечатку файла результатов – 2–3 примера.

Высылаемый на проверку преподавателю архив должен содержать отчёт и все файлы программы – все модули, exe-модуль, файлы данных, файлы результатов.

**Внимание!!!**

**Необходимо предусмотреть обработку ошибок.   
Никакие входные данные не должны нарушать работу программы!! Программа не должна «зависать» или прекращать выполнение по неизвестной причине – обязательна выдача соответствующей диагностики.**

При выполнении задания следует использовать средства объектно-ориентированного программирования. Выбор конкретного средства разработки оставляется за студентом. При наличии определённых требований к программному обеспечению студенту следует уведомлять об этом преподавателя.

Рекомендуется при разработке программного средства использовать материалы лабораторных работ (в зависимости от темы задания).

Программа должна управляться посредством меню, в котором должны присутствовать следующие пункты: "Автор", "Тема" (с полной информацией о разработчике и теме задания), "Данные" (выбор способа задания исходных данных – чтение из файла или ввод с клавиатуры), "Расчёты", "Запись результатов в файл" – и другие, определяемые конкретным заданием. При вводе данных с клавиатуры необходимо использовать соответствующую форму, а также предусмотреть возможность вызова справки с примером формата данных. При чтении из файла – должна открываться своя папка. Все результаты расчётов должны отображаться на экране и выводиться в файл (по требованию пользователя). При введении автором каких-либо ограничений (размер алфавита и т.п.) они должны быть описаны в пояснительной записке и в соответствующем пункте меню.

## Вариант 1

Написать программу для автоматического построения детерминированного конечного автомата (ДКА) по словесному описанию языка.

Вход программы: алфавит языка, обязательная начальная подцепочка, кратность длины всех цепочек языка, цепочки для распознавания.

Выход: построенный ДКА (все 5 элементов), результат проверки цепочек.

Подробно:

Язык задан своим алфавитом, обязательной начальной цепочкой и указанием кратности длины всех цепочек языка. В начальной цепочке не должно находиться символов, не содержащихся в алфавите. В крайнем случае она может быть и пустой. Кратность длины цепочек задаётся любым натуральным числом. Кратность, равная единице, равносильна отсутствию ограничений на длину цепочек.

Программа должна:

1. по предложенному описанию регулярного языка строить ДКА, распознающий этот язык, в том виде, как он рассматривался в теории, раздел 2.2.2;
2. с помощью построенного ДКА проверять вводимые пользователем цепочки на их принадлежность этому языку.

ДКА может быть полностью или неполностью определённым – по желанию разработчика. Функция переходов ДКА может изображаться в виде таблицы и графа, вариант вида отображения должен выбираться в меню.

После построения ДКА пользователь может вводить произвольные цепочки для проверки их на принадлежность исходному языку. Разбор цепочек автоматом следует поэтапно отображать на экране в виде последовательной смены конфигураций в соответствии с лабораторной работой №2.

Рассмотрим пример построения ДКА.

Задан язык: алфавит {a,b,c}, обязательная начальная цепочка ‘aab’, длина цепочек кратна 2. Анализируем задание: язык будет состоять из цепочек чётной длины, начинающихся с ‘aab’, например {aaba, aabb, aabc, aabaaa, aabaab, aabccc,…}. Тогда ДКА должен иметь вид M(Q,{a,b,с},δ,q0,F), множество состояний Q и заключительные состояния F определятся в процессе построения. Разберёмся с построением функции переходов. Очевидно, что пустая цепочка в языке не содержится (поскольку есть непустая обязательная начальная цепочка). Сначала определимся с минимальной цепочкой языка и построим для неё граф переходов. Это ‘aab’ и еще один (любой) символ алфавита.

q0

q1

q4

b

a

q2

q3

a

a,b,c

После этой цепочки допускается её наращивание парами любых символов алфавита. Возврат из последнего состояния в q3 по любому символу алфавита позволит добавлять такие пары символов. Функция переходов δ построена:

q0

q1

q4

b

a

q2

q3

a

a,b,c

a,b,c

Q={q0,q1,q2,q3,q4}, F={q4}.

В табличном виде она представляется очевидным образом.

## Вариант 2

Написать программу для автоматического построения регулярной грамматики (леволинейной или праволинейной) по словесному описанию языка.

Вход программы: алфавит языка, обязательные начальная и конечная подцепочки, тип грамматики (ЛЛ либо ПЛ), 2 числа – диапазон длин для генерации цепочек.

Выход: построенная грамматика (все 4 элемента), результат генерации цепочек.

Подробно:

Язык задан своим алфавитом, обязательной начальной и конечной подцепочками, которые должны присутствовать во всех цепочках языка. В заданных подцепочках не должно находиться символов, не содержащихся в алфавите. В крайнем случае они (одна или обе) могут быть и пустыми. Особый случай – это пересечение заданных цепочек по одному или нескольким символам, в том числе и полное их совпадение.

Программа должна:

1. по предложенному описанию регулярного языка строить регулярную грамматику, генерирующую этот язык, в том виде, как она рассматривалась в теории, раздел 1.3.1;
2. с помощью построенной грамматики генерировать все цепочки языка в заданном пользователем диапазоне длин.

Грамматика должна строиться и праволинейная, и леволинейная. Желаемый тип грамматики выбирается пользователем в меню. Все правила грамматики должны соответствовать выбранному типу! Отдельно следует указывать, какой нетерминальный символ является целевым. Если в грамматике используется пустое правило, то необходимо дать пояснение, каким именно символом обозначается пустая цепочка.

После построения грамматики пользователь может убедиться в её правильности путём генерации всех цепочек языка в том диапазоне длин, который он задаст. Генерацию каждой цепочки языка следует поэтапно отображать на экране в виде цепочки вывода (в соответствии с примерами раздела 1.4.1.). Генерация осуществляется в соответствии с лабораторной работой №1.

Рассмотрим пример построения регулярной грамматики.

Задан язык: алфавит {a,b,c}, обязательная начальная цепочка ‘саа’, конечная цепочка ‘aab’. Анализируем задание: язык будет состоять из цепочек любой длины, начинающихся с цепочки ‘cаа’ и заканчивающихся на ‘aab’, например {сaab, саaab, сaaааb, caaсааb, сaabaab, …}.

Для определённости выберем праволинейную грамматику (ПЛ). В правой части её правил вывода может быть только один нетерминал, и он должен располагаться справа от цепочек терминальных символов.

Все цепочки языка должны начинаться с заданной цепочки ‘саа’, после которой может находиться всё, что угодно. Значит, правило для целевого символа будет иметь вид: S→caaА, а для нетерминала А: А→aA|bA|cA. В конце должна быть ‘aab’, значит, выход из нетерминала А будет по ней: А→aA|bA|cA|aab. Осталось учесть возможные пересечения цепочек, это ‘сaab’, ‘саaab’. Их нужно выводить из целевого символа. Итак, грамматика имеет вид: G({a,b,c},{S,A},P,S), Р: S→caaА|сaab|саaab; А→aA|bA|cA|aab.

## Вариант 3

Написать программу для автоматического построения регулярного выражения (РВ) по словесному описанию языка.

Вход программы: алфавит языка, обязательная начальная подцепочка, выбранный символ алфавита, его кратность (натуральное число), 2 числа – диапазон длин для генерации цепочек.

Выход: построенное регулярное выражение, результат генерации цепочек.

Подробно:

Язык задан своим алфавитом, обязательной начальной цепочкой и указанием кратности вхождений некоторого символа во все цепочки языка. В начальной цепочке не должно находиться символов, не содержащихся в алфавите. В крайнем случае она может быть и пустой. Кратность числа вхождений выбранного символа задаётся любым натуральным числом. Кратность, равная единице, равносильна отсутствию ограничений на вхождения данного символа, т.е. он может встречаться в цепочках любое количество раз. При любом значении кратности количество вхождений символа в цепочку может быть и нулевым.

Программа должна:

1. по предложенному описанию регулярного языка строить РВ, генерирующее этот язык, в том виде, как было рассмотрено в теории, раздел 2.1.1;
2. с помощью построенного РВ генерировать все цепочки языка в заданном пользователем диапазоне длин.

При генерации цепочек языка по РВ можно использовать любые структуры и алгоритмы, в том числе и прямой перебор.

Рассмотрим пример построения регулярного выражения.

Задан язык: алфавит {0,1,a,b}, обязательная начальная цепочка ‘01a’, количество вхождений символа ‘а’ кратно 2. Анализируем задание: язык будет состоять из цепочек с чётным количеством символов ‘а’, начинающихся с ‘01a’, например {01аa, 01аba, 01аab, 01аac, 01аaaa,…}. Для начала определим минимально возможную цепочку этого языка. Это ‘01aa’ – она удовлетворяет всем условиям и является при этом самой короткой. Это основа нашего РВ.

Удлинить эту цепочку можно, поместив после неё или между цепочкой ’01a’ и ‘a’ в ней любое количество любых символов алфавита, за исключением символа ‘a’ – для него нужно подсчитывать количество, поэтому его следует 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 01a(0+1+b)\*a(0+1+b)\*. Но по построенному таким образом выражению нельзя получить много символов ‘a’. Для увеличения количества ‘a’ можно добавить после построенного выражения (aa)\* для сохранения чётности. Но тогда 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. Итоговое регулярное выражение будет иметь вид:   
01a(0+1+b)\*a(0+1+b)\*((0+1+b)\*а(0+1+b)\*a(0+1+b))\*. Можно несколько упростить построенное выражение, убрав одну из скобок (0+1+b)\* – там, где она повторяется. Окончательный ответ: 01a(0+1+b)\*a(0+1+b)\*(а(0+1+b)\*a(0+1+b))\***.**

## Вариант 4

Написать программу для автоматического построения детерминированного конечного автомата (ДКА), эквивалентного заданной регулярной грамматике.

Вход программы: терминальный и нетерминальный алфавиты грамматики, целевой символ, правила грамматики, цепочки для распознавания.

Выход: построенный ДКА (все 5 элементов), результат проверки цепочек.

Подробно:

Язык задан регулярной грамматикой, причём она может быть не автоматного вида. При написании программы разработчику разрешается выбрать один из двух типов регулярной грамматики (ЛЛ или ПЛ) и следует информировать об этом пользователя. Терминальный алфавит грамматики может включать в себя любые символы, в нетерминальном алфавите могут использоваться заглавные буквы латинского алфавита или (на усмотрение разработчика) слова. Правила задаваемой грамматики должны соответствовать выбранному типу. Для того чтобы в исходной грамматике можно было использовать пустое правило, необходимо предусмотреть поле ввода для символа, которым пользователь может обозначить пустую цепочку.

Программа должна:

1. по заданной регулярной грамматике строить эквивалентный ДКА, распознающий этот же язык, в том виде, как он рассматривался в теории, раздел 2.2.2;
2. с помощью построенного ДКА проверять вводимые пользователем цепочки на их принадлежность этому языку.

ДКА должен распознавать язык, задаваемый исходной грамматикой, т.е. являться эквивалентной конструкцией. Функция переходов ДКА может изображаться в виде таблицы или графа, вариант вида её представления выбирается разработчиком. Для удобства построения автомата рекомендуется предварительно привести заданную грамматику к автоматному виду (в соответствии с лекционным разделом 2.2.1).

При выборе такого способа построения ДКА, когда сначала по заданной грамматике строится эквивалентный НКА, а затем он приводится к детерминированному виду, промежуточный результат в виде НКА необходимо также отображать на экране по просьбе пользователя.

После построения ДКА пользователь может вводить произвольные цепочки для проверки их на принадлежность исходному языку. Разбор цепочек автоматом следует поэтапно отображать на экране в виде последовательной смены конфигураций в соответствии с лабораторной работой №2.

Рассмотрим пример построения ДКА.

Язык задан праволинейной грамматикой: G({a,b,c},{S,A},P,S),   
Р: S→aA|bA|cA; А→aS|bS|cS|aab.

Сначала следует привести грамматику к автоматному виду, добавив 2 нетерминала для разделения на символы цепочки ‘aab’. Правила примут вид:   
Р`: S→aA|bA|cA; А→aS|bS|cS|aB; B→aC; C→b. Теперь нужно по правилам грамматики построить конечный автомат, в котором состояния будут получены из нетерминалов грамматики, а переходы определяются терминальными символами. Заключительным будет являться то состояние, в которое происходит переход по одному символу (C→b), следовательно, придётся добавить ещё одно состояние D. Функцию переходов сначала будем строить графически:

Как видно из графа переходов, автомат получился недетерминированный (из состояния А есть два перехода по символу ‘a’). Далее можно применить алгоритм преобразования НКА к ДКА, рассмотренный подробно в лекциях, раздел 2.2.2.

S

A

D

a

a,b,c

B

C

a

b

a,b,c

Построим таблицу переходов нашего автомата и преобразуем его в ДКА согласно вышеупомянутому алгоритму.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | ***вход*** | | | Исходную таблицу переходов отделим от остальной части жирной линией.  Для упрощения процесса будем создавать не все возможные сочетания исходных состояний, а только те, которые реально возникают при построении. Сначала занесём в таблицу SB. Затем появляются AC, SD. Состояния исходного автомата B, C, D (выделены синим) оказались недостижимыми. Удалим их. |
| ***состояние*** | | *а* | *b* | *c* |
| S | | {A} | {A} | {A} |
| A | | {S,B} | {S} | {S} |
| B | | {C} | – | – |
| C | | – | {D} | – |
| D | | – | – | – |
| SB | B | {AC} | {A} | {A} |
| AC | C | {SB} | {SD} | {S} |
| SD | D | {A} | {A} | {A} |

Новые состояния для удобства переобозначим B, C, D. Заключительным состоянием станет состояние D (поскольку в состояние SD входит то состояние D, которое было заключительным в исходном автомате). Новая таблица переходов примет следующий вид:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***вход*** | | | Построение графа переходов по таблице:  b,c  S  A  D  a  a,b,c  B  C  a  b  b,c  a  c  a,b,c |
| ***состояние*** | *a* | *b* | *c* |
| S | {A} | {A} | {A} |
| A | {B} | {S} | {S} |
| B | {C} | {A} | {A} |
| C | {B} | {D} | {S} |
| D | {A} | {A} | {A} |

Итак, ДКА построен и имеет вид: M({S,A,B,C,D},{a,b,с},δ,S,{D}), где функция переходов δ получена выше.

## Вариант 5

Написать программу для автоматического построения регулярной грамматики (леволинейной или праволинейной), эквивалентной заданному конечному автомату (КА).

Вход программы: КА в виде: алфавит языка, множество состояний, начальное состояние, множество заключительных состояний, функция переходов в табличном виде; тип грамматики (ЛЛ либо ПЛ), 2 числа – диапазон длин для генерации цепочек.

Выход: построенная грамматика (все 4 элемента), результат генерации цепочек.

Подробно:

Язык задан конечным автоматом. Автомат может быть полностью или неполностью определённым, детерминированным или недетерминированным, заключительных состояний может быть несколько.

Программа должна:

1. по предложенному распознавателю регулярного языка в виде КА строить регулярную грамматику, генерирующую этот язык, в том виде, как она рассматривалась в теории, раздел 1.3.1;
2. с помощью построенной грамматики генерировать все цепочки языка в заданном пользователем диапазоне длин.

Грамматика должна строиться и праволинейная, и леволинейная. Желаемый тип грамматики выбирается пользователем в меню. Все правила грамматики должны соответствовать выбранному типу! Отдельно следует указывать, какой нетерминальный символ является целевым. Если в грамматике используется пустое правило, то необходимо дать пояснение, каким именно символом обозначается пустая цепочка.

После построения грамматики пользователь может убедиться в её правильности путём генерации всех цепочек языка в том диапазоне длин, который он задаст. Генерацию каждой цепочки языка следует поэтапно отображать на экране в виде цепочки вывода (в соответствии с примерами раздела 1.4.1.). Генерация осуществляется в соответствии с лабораторной работой №1.

Рассмотрим пример построения регулярной грамматики.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***вход*** | | | Задан ДКА M({S,A,B},{0,1,2},δ,S,{B}), функция переходов δ представлена в табличном виде. Построить праволинейную грамматику. В правой части её правил вывода может быть только один нетерминал, и он должен располагаться справа от терминальных символов. |
| ***состояние*** | 0 | 1 | 2 |
| S | {S} | {A} | – |
| A | {B} | – | {B} |
| B | {A} | – | {A} |

Нетерминалами будут являться состояния ДКА: {S,A,B}, заключительному состоянию должно соответствовать пустое правило. Целевым символом будет нетерминал, соответствующий начальному состоянию исходного ДКА. Правила грамматики строятся по строкам таблицы переходов следующим образом: если Х∈δ(Y,t), X,Y∈Q, t – входной символ, то соответствующее правило праволинейной грамматики будет иметь вид Y → tX.

Итак, грамматика имеет вид: G({0,1,2},{S,A,B},P,S), Р: S→0S|1А; A→0B|2B; B→0A|2A|λ.

## Вариант 6

Написать программу для автоматического построения грамматики, эквивалентной заданному регулярному выражению (РВ).

Вход программы: регулярное выражение в виде строки символов, 2 числа – диапазон длин для генерации цепочек.

Выход: построенная грамматика (все 4 элемента), результат генерации цепочек.

Подробно:

Язык задан регулярным выражением. При его записи могут быть использованы символы алфавита языка, а также: «+» (выбор одного из слагаемых), круглые скобки, «\*» для обозначения итерации.

Программа должна:

1. по предложенному регулярному выражению строить эквивалентную грамматику, генерирующую этот же язык, в том виде, как она рассматривалась в теории, раздел 1.3.1;
2. с помощью построенной грамматики генерировать все цепочки языка в заданном пользователем диапазоне длин.

Грамматика может строиться любая – контекстно-свободная или регулярная, по выбору разработчика. Отдельно следует указывать, какой нетерминальный символ является целевым. Если в грамматике используется пустое правило, то необходимо дать пояснение, каким именно символом обозначается пустая цепочка.

После построения грамматики пользователь может убедиться в её правильности путём генерации всех цепочек языка в том диапазоне длин, который он задаст. Генерацию каждой цепочки языка следует поэтапно отображать на экране в виде цепочки вывода (в соответствии с примерами раздела 1.4.1.). Генерация осуществляется в соответствии с лабораторной работой №1.

Рассмотрим пример построения КС-грамматики.

Задано регулярное выражение: ((0+1+b)\*a(0+1+b)\*a)\*(0+1+b)\*a(0+1+b)\*01a.

Для построения правил грамматики следует сделать разбор исходного регулярного выражения. Каждая скобка обозначается своим нетерминалом. Если на скобке стоит звёздочка (итерация), значит, на этом нетерминале будет явная рекурсия и пустое правило. Если в выражении стоит «+», то это означает альтернативу в правилах.

Обозначим первую большую скобку через A=((0+1+b)\*a(0+1+b)\*a)\*, вторую B=(0+1+b)\*. Само РВ должно порождаться из целевого символа грамматики. Тогда первое правило будет иметь вид: S→ABaB01a. В правиле для A будет присутствовать B: A→BaBa. Поскольку на скобке есть звёздочка, то 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.

Аналогично строится регулярная грамматика, только там следует учитывать, что в правой части правил может использоваться не более одного нетерминала, и располагаться во всех правилах грамматики он должен с одной стороны от цепочки терминальных символов.

## Вариант 7

Написать программу для автоматического приведения заданной контекстно-свободной грамматики (КС-грамматики) к нормальной форме Хомского (БНФ).

Вход программы: терминальный и нетерминальный алфавиты грамматики, целевой символ, правила грамматики, 2 числа – диапазон длин для генерации цепочек.

Выход: построенная грамматика в БНФ (все 4 элемента), результат генерации цепочек по обеим грамматикам.

Подробно:

Язык задан КС-грамматикой, причём для приведения к БНФ она должна находиться в каноническом виде (раздел лекций 3.2.2). Приводить её к этому виду не требуется, достаточно только проверить корректность задания – действительно ли исходная грамматика находится в каноническом виде – и при отрицательном результате выдать соответствующее сообщение. Причём в этом сообщении должны быть конкретно указаны причины, почему именно грамматика не имеет канонического вида (например: «в грамматике присутствуют цепные правила: А→С», или «в грамматике присутствует недостижимый символ: В»). Для того чтобы в исходной грамматике можно было использовать пустое правило, необходимо либо предусмотреть поле ввода для символа, которым обозначается пустая цепочка, либо дать пояснения пользователю, как именно ему следует задавать пустое правило.

Программа должна:

1. проверить заданную КС-грамматику – находится ли она в каноническом виде и при отрицательном результате выдать сообщение;
2. привести заданную КС-грамматику к нормальной форме Хомского (раздел 3.3.1);
3. проверить построенную грамматику (БНФ) на эквивалентность исходной.

Для проверки построенной грамматики в БНФ на эквивалентность исходной по обеим грамматикам следует сгенерировать множества всех цепочек в заданном пользователем диапазоне длин и проверить эти множества на идентичность. Генерация цепочек осуществляется в соответствии с лабораторной работой №1. При обнаружении несовпадения должна выдаваться диагностика различий – где именно несовпадения и в чём они состоят. Для удобства сравнения множества цепочек необходимо упорядочить, цепочки перенумеровать, для генерации цепочек по каждой грамматике сделать отдельную кнопку. Следует предусмотреть возможность многократного изменения диапазона длин цепочек и повторной их генерации для новых значений длины.

Приведение КС-грамматики к нормальной форме Хомского следует осуществлять согласно алгоритму, изложенному в лекционном материале (раздел 3.3.1).

## Вариант 8

Написать программу для автоматического построения детерминированного конечного автомата (ДКА) по словесному описанию языка.

Вход программы: алфавит языка, обязательная конечная подцепочка, цепочки для распознавания.

Выход: построенный ДКА (все 5 элементов), результат проверки цепочек.

Подробно:

Язык задан своим алфавитом и обязательной конечной подцепочкой всех цепочек языка. В конечной подцепочке не должно находиться символов, не содержащихся в алфавите. В крайнем случае она может быть и пустой.

Программа должна:

1. по предложенному описанию регулярного языка строить ДКА, распознающий этот язык, в том виде, как он рассматривался в теории, раздел 2.2.2;
2. с помощью построенного ДКА проверять вводимые пользователем цепочки на их принадлежность этому языку.

ДКА должен быть полностью определённым. Функция переходов ДКА может изображаться в виде таблицы или графа, вариант вида её представления выбирается разработчиком.

Наиболее простой способ построения такого ДКА состоит в том, чтобы сначала по описанию языка построить НКА (недетерминированный конечный автомат), а затем преобразовать его согласно рассмотренному в разделе 2.2.2 алгоритму. При выборе такого способа построения ДКА промежуточный результат в виде НКА необходимо также отображать на экране по просьбе пользователя.

По желанию автора допускаются и другие способы построения ДКА.

После построения ДКА пользователь может вводить произвольные цепочки для проверки их на принадлежность исходному языку. Разбор цепочек автоматом следует поэтапно отображать на экране в виде последовательной смены конфигураций в соответствии с лабораторной работой №2.

Рассмотрим пример построения ДКА.

Задан язык: алфавит {a,b,c}, обязательная конечная подцепочка ‘aaba’. Анализируем задание: язык будет состоять из цепочек любой длины, заканчивающихся на ‘aaba’, например {aaba, аaaba, baaba, caaba, aaaaba, …}. Тогда ДКА должен иметь вид M(Q,{a,b,с},δ,q0,F), множество состояний Q и заключительные состояния F определятся в процессе построения. Разберёмся с построением функции переходов δ. Очевидно, что пустая цепочка в языке не содержится (поскольку есть непустая обязательная конечная цепочка). Сначала определимся с минимальной цепочкой языка – это ‘aaba’, и построим для неё граф переходов.

q0

q1

q4

b

a

q2

q3

а

a

Если выбрать способ с предварительным построением НКА, то такой автомат выглядит очевидным образом. Сначала могут быть прочитаны любые символы алфавита в любом количестве, а затем конечная подцепочка:

q0

q1

q4

b

a

а,b,с

q2

q3

а

a

Недетерминированность автомата вызвана тем, что из начального состояния существует два перехода по одному символу алфавита (‘a’). Осталось преобразовать построенный автомат в детерминированный. Для этого построим таблицу переходов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | ***вход*** | | | Исходную таблицу переходов отделим от остальной части жирной линией.  Для упрощения процесса будем создавать не все возможные новые состояния, которые могут получиться в результате сочетаний исходных состояний, а только те, которые реально возникают при построении. Сначала это единственное состояние q0q1 – занесём его в таблицу. Затем последовательно появятся q0q1q2, q0q3, q0q1q4. Все состояния исходного автомата, кроме q0, оказались недостижимыми. В таблице они выделены синим цветом. Удалим их. |
| ***состояние*** | | a | b | c |
| q0 | | {q0,q1} | {q0} | {q0} |
| q1 | | {q2} | – | – |
| q2 | | – | {q3} | – |
| q3 | | {q4} | – | – |
| q4 | | – | – | – |
| q0q1 | A | {q0q1q2} | {q0} | {q0} |
| q0q1q2 | B | {q0q1q2} | {q0q3} | {q0} |
| q0q3 | C | {q0q1q4} | {q0} | {q0} |
| q0q1q4 | D | {q0q1q2} | {q0} | {q0} |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***вход*** | | | Новые состояния для удобства переобозначим A, B, C, D. Заключительными состояниями станут те, которые содержат q4. Здесь такое состояние одно – D. Новая таблица переходов представлена слева:  c  a  q0  A  D  b  a  b,с  B  C  a  a  b,c  a  b,c  b,c |
| ***состояние*** | a | b | c |
| q0 | {A} | {q0} | {q0} |
| A | {B} | {q0} | {q0} |
| B | {B} | {C} | {q0} |
| C | {D} | {q0} | {q0} |
| D | {B} | {q0} | {q0} |

Граф переходов построен по таблице:

Q={q0,A,B,С,D }, F={D}.

ДКА построен.

## Вариант 9

Написать программу для автоматического построения регулярного выражения (РВ) по словесному описанию языка.

Вход программы: алфавит языка, обязательные начальная и конечная подцепочки, кратность длины всех цепочек языка, 2 числа – диапазон длин для генерации цепочек.

Выход: построенное регулярное выражение, результат генерации цепочек.

Подробно:

Язык задан своим алфавитом, обязательной начальной и конечной подцепочками и указанием кратности длины всех цепочек языка. В заданных подцепочках не должно находиться символов, не содержащихся в алфавите. В крайнем случае они (одна или обе) могут быть и пустыми. Особый случай – это пересечение заданных цепочек по одному или нескольким символам, в том числе и полное их совпадение. Кратность длины цепочек задается любым натуральным числом. Кратность, равная единице, равносильна отсутствию ограничений на длину цепочек.

Программа должна:

1. по предложенному описанию регулярного языка строить РВ, задающее этот язык, в том виде, как было рассмотрено в теории, раздел 2.1.1;
2. с помощью построенного РВ генерировать все цепочки языка в заданном пользователем диапазоне длин.

При генерации цепочек языка по РВ можно использовать любые структуры и алгоритмы, в том числе и прямой перебор.

Рассмотрим пример построения регулярного выражения.

Задан язык: алфавит {a,b,c}, обязательная начальная цепочка ‘саа’, конечная цепочка ‘aab’, длина цепочек кратна 2. Анализируем задание: язык будет состоять из цепочек чётной длины, начинающихся с цепочки ‘cаа’ и заканчивающихся на ‘aab’, например {сaab, сaaааb, caaасааb, сaabсaab, …}.

Для начала определим минимально возможную цепочку этого языка. Это ‘сaab’ – она получается при максимальном пересечении начальной и конечной цепочек, удовлетворяет всем условиям и является при этом самой короткой. Удлинить её можно путём уменьшения области пересечения начальной и конечной цепочек. Следующая по длине цепочка могла бы быть ‘сaаab’ (пересечение по одному символу), но она не удовлетворяет условию чётности длины. А вот ещё следующая цепочка – ‘сaaааb’ – получена простым соединением начальной и конечной цепочек. Это основа нашего РВ. Её можно удлинять, помещая между начальной и конечной цепочками некоторое количество символов алфавита, причём для сохранения чётности длины это количество тоже должно быть чётным. Для генерации любого количества пар символов используется итерация, а выбор одного из нескольких символов записывается в виде суммы.

Итоговое регулярное выражение будет иметь вид: саа((а+b+c)(a+b+c))\*aab+caaaab.

Если в этом же задании изменить кратность длины на единицу, то итоговое РВ изменится таким образом: добавлять можно будет не пары символов, а по одному символу любое количество раз, и пересечения цепочек будут использованы все. Итоговое регулярное выражение будет иметь вид: саа(a+b+c)\*aab+caaab +caaaab.

## Вариант 10

Написать программу для автоматического построения регулярной грамматики (леволинейной или праволинейной) по словесному описанию языка.

Вход программы: алфавит языка, обязательная конечная подцепочка, кратность длины всех цепочек языка, тип грамматики (ЛЛ либо ПЛ), 2 числа – диапазон длин для генерации цепочек.

Выход: построенная грамматика (все 4 элемента), результат генерации цепочек.

Подробно:

Язык задан своим алфавитом, обязательной конечной подцепочкой, которая должна присутствовать во всех цепочках языка, и указанием кратности длины всех цепочек языка. В конечной цепочке не должно находиться символов, не содержащихся в алфавите. В крайнем случае она может быть и пустой. Кратность длины цепочек задаётся любым натуральным числом. Кратность, равная единице, равносильна отсутствию ограничений на длину цепочек.

Программа должна:

1. по предложенному описанию регулярного языка строить регулярную грамматику, генерирующую этот язык, в том виде, как она рассматривалась в теории, раздел 1.3.1;
2. с помощью построенной грамматики генерировать все цепочки языка в заданном пользователем диапазоне длин.

Грамматика должна строиться и праволинейная, и леволинейная. Желаемый тип грамматики выбирается пользователем в меню. Все правила грамматики должны соответствовать выбранному типу! Отдельно следует указывать, какой нетерминальный символ является целевым. Если в грамматике используется пустое правило, то необходимо дать пояснение, каким именно символом обозначается пустая цепочка.

После построения грамматики пользователь может убедиться в её правильности путём генерации всех цепочек языка в том диапазоне длин, который он задаст. Генерацию каждой цепочки языка следует поэтапно отображать на экране в виде цепочки вывода (в соответствии с примерами раздела 1.4.1.). Генерация осуществляется в соответствии с лабораторной работой №1.

Рассмотрим пример построения регулярной грамматики.

Задан язык: алфавит {a,b,c}, обязательная конечная цепочка ‘aab’, кратность длины 2. Анализируем задание: язык будет состоять из цепочек чётной длины, заканчивающихся на ‘aab’, например {сaab, аaab, bааb, caaааb, сabaab, …}.

Для примера выберем праволинейную грамматику (ПЛ). В правой части её правил вывода может быть только один нетерминал, и он должен располагаться справа от цепочек терминальных символов. Для соблюдения кратности длины цепочек будет использоваться чередование нетерминалов. Длина кратна двум, следовательно, потребуется два нетерминала: S→aA|bA|cA, А→aS|bS|cS. Выход должен происходить по обязательной конечной цепочке ‘aab’, в силу чётности длины цепочек перед ней должен быть как минимум один символ. Следовательно, эта цепочка должна порождаться нетерминалом А: А→aab.

Итак, построенная грамматика имеет вид:   
G({a,b,c},{S,A},P,S), Р: S→aA|bA|cA; А→aS|bS|cS|aab.

Задание на лабораторные работы

Лабораторные работы основаны на лекционном материале; каждая выполняется после изучения соответствующего теоретического раздела. До выполнения лабораторной работы нужно внимательно разобраться с примерами, ответить на контрольные вопросы изученного теоретического раздела, а также решить задачи, предлагаемые в составе контрольных вопросов.

Каждая работа снабжена методическими указаниями, сопровождающими текст задания. Рекомендуется внимательно читать задание и выполнять работу в строгом соответствии с требованиями.

Среда программирования – любая, по желанию студента. Рекомендуется использовать Delphi, C# или аналоги.

**Внимание!   
*При выполнении лабораторных работ необходимо предусматривать обработку любых возможных ошибок ввода. Программа не должна «зависать» или вести себя иным некорректным образом ни при каких начальных данных! При вводе неправильных начальных данных должно быть выведено сообщение об ошибке пользователя и предложено повторить ввод правильно (как вариант – можно программно запрещать ввод неправильных данных).***

По каждой лабораторной работе необходимо выполнять отчёт, включающий в себя:

1. постановку задачи;
2. описание входных данных программы и её результатов;
3. описание основных переменных, а также основных блоков и подпрограмм;
4. алгоритм решения задачи;
5. текст программы;
6. результаты работы, если они есть (вариант – копия экрана), или результаты тестирования;
7. ответы на контрольные вопросы.

***Преподавателю на проверку необходимо предоставлять отчёт, исходный код и откомпилированный модуль (exe-файл).***

Для получения зачёта по каждой лабораторной работе студентом должно быть выполнено основное задание. Дополнительная часть не обязательна, она предназначена для желающих более полно и глубоко изучить предмет.

Лабораторная работа № 1 Генерация цепочек языка

Пусть язык задан контекстно-свободной грамматикой (теоретический материал разделов 1.1–1.4). Написать программу, которая по заданной грамматике будет генерировать ВСЕ цепочки языка в некотором диапазоне длин. Использовать только левосторонний или правосторонний вывод! Диапазон длин генерируемых цепочек должен задаваться пользователем при запуске программы.

Предусмотреть возможность выбора пользователю – использовать заданную в программе грамматику или вводить свою с клавиатуры.

На вход программы подаётся КС-грамматика (терминальный и нетерминальный алфавиты, целевой символ, правила вывода), задаётся диапазон длин цепочек, указывается тип вывода (левосторонний или правосторонний).

Рекомендуется для ввода исходных данных использовать соответствующую форму. При вводе правил грамматики не рекомендуется предоставлять пользователю излишнюю свободу действий, например, удобнее запретить ввод в левой части правил более чем одного нетерминального символа, чем проверять введённые правила на соответствие контекстно-свободному типу.

На выходе должен быть список построенных цепочек. Все цепочки в списке должны быть различны. При получении одинаковых цепочек (разными способами) их следует сохранять в списке выводимых цепочек только в случае выполнения дополнительного задания.

Проверить программу на примерах из лекционного курса и на заданиях из контрольных вопросов к теоретической части.

Дополнительно: Дополнить предыдущую программу таким образом, чтобы для одной или нескольких цепочек (цепочки выбирает пользователь из числа построенных на предыдущем этапе работы) строилось дерево вывода, т.е. была структурно представлена последовательность правил, использованных при построении цепочки.

***Контрольные вопросы:***

1. Как поведёт себя программа, если при вводе правил грамматики сделать ошибку – ввести правила грамматики, не соответствующей КС-типу, или не использовать какие-то символы исходных алфавитов, или поместить в левой части правил терминальный символ?
2. Используется ли при генерации цепочек рекурсия? Если да, то как ограничена её глубина?
3. Какой язык порождается грамматикой G({0},{S,A},{S→AAA, A→AAA |0}, S)? Правильно ли Ваша программа работает на этом примере? Если нет – доработайте программу.
4. Разрешает ли Ваша программа использовать в исходной грамматике пустые правила? Если нет – доработайте программу.

Лабораторная работа № 2 Моделирование работы ДКА

Пусть регулярный язык задаётся конечным автоматом – ДКА (теоретический материал разделов 1.5, 2.2). Написать программу, которая будет проверять по заданному автомату вводимую цепочку и делать вывод о том, принадлежит ли она рассматриваемому регулярному языку. В случае отрицательного ответа необходимо давать пояснение, по какой причине цепочка не принадлежит языку – например, «в цепочке присутствуют посторонние символы», «после прочтения цепочки автомат не пришёл в конечное состояние» и т.п. Исходный автомат вводить с клавиатуры в соответствии с определённым форматом. Ввод цепочек также производить с клавиатуры.

На вход программы подаётся ДКА (множество состояний, алфавит языка, начальное состояние, множество заключительных состояний, функция переходов в виде таблицы) и проверяемая цепочка символов (может вводиться многократно, т.е. возможно проверить любое количество цепочек). При этом в проверяемую цепочку могут входить и символы, не принадлежащие алфавиту языка; цепочка может быть и пустой.

Программа должна предоставлять пользователю возможность изменять начальное и конечные состояния с сохранением введённой функции переходов для заданного автомата.

Выход – отображение на экране процесса проверки цепочки в виде последовательной смены конфигураций и вывод результата – сообщения, принадлежит ли цепочка языку.

Проверить работу программы на примерах из лекций и задачах из контрольных вопросов.

Дополнительно:

1. Предоставить пользователю возможность не только вводить данные с клавиатуры, но и загружать автомат из файла (выбор – в соответствующем пункте меню или нажатием кнопки в исходном окне программы). При этом следует накладывать определённые ограничения на формат файла и производить соответствующие проверки во избежание загрузки некорректных данных.
2. Также по желанию пользователя результаты помимо вывода на экран сохранять в файле. Выбор – аналогично загрузке данных.

***Контрольные вопросы:***

* 1. Как поведёт себя программа, если при вводе таблицы переходов ДКА сделать (случайно или преднамеренно) ошибку – например, ввести несуществующее состояние?
  2. Все ли ячейки таблицы переходов исходного ДКА обязательно должны быть заполнены или можно использовать неполностью определённый ДКА?
  3. В каком случае ДКА распознаёт пустую цепочку как цепочку языка?

Лабораторная работа № 3 Моделирование работы МПА

Пусть контекстно-свободный язык задаётся детерминированным автоматом с магазинной памятью – ДМПА (теоретический материал раздела 3.1). Написать программу, которая будет проверять для вводимой цепочки, принадлежит ли она заданному КС-языку. В случае отрицательного ответа необходимо давать пояснение, по какой причине цепочка не принадлежит языку (аналогично лаб. раб №2) Исходный автомат вводить с клавиатуры в соответствии с определённым форматом. Ввод цепочек также производить с клавиатуры, выполняя его до тех пор, пока не возникнет желание закончить работу.

На вход программы подаётся ДМПА (множество состояний, алфавит языка, алфавит магазина, начальное состояние, начальное содержимое стека, множество заключительных состояний, функция переходов в виде списка правил) и проверяемая цепочка символов (может вводиться многократно, т.е. возможно проверить любое количество цепочек). При этом в проверяемую цепочку могут входить и символы, не принадлежащие алфавиту языка; цепочка может быть и пустой.

Рекомендуется для ввода исходных данных использовать соответствующую форму с полями выбора. При вводе функции переходов не рекомендуется предоставлять пользователю излишнюю свободу действий.

Выход – отображение на экране процесса проверки цепочки в виде последовательной смены конфигураций и вывод результата – сообщения, принадлежит ли цепочка языку.

Внимание!!! В ходе проверки цепочки ДМПА может проделывать пустые такты не только после прочтения всей цепочки, но и в её середине. Внимательно прочтите соответствующий раздел лекций 3.1.

Проверить работу программы на примерах из лекций и задачах из контрольных вопросов.

Дополнительно: Предоставить пользователю возможность вносить некоторые изменения в исходные данные без полного их обновления.

***Контрольные вопросы:***

1. Обязательно ли должна быть задана функция переходов для всех возможных сочетаний «состояние, входной символ, символ стека»? Как поведёт себя автомат, если при проверке цепочки встретит неизвестную комбинацию?
2. В каком случае цепочка считается допущенной автоматом, а в каком – нет?
3. Возможно ли в качестве алфавита магазина использовать символы, отличные от алфавита языка? Изменится ли работа программы, если изменить только алфавит магазинных символов (и соответствующие правила функции переходов)? Если в Вашей программе это недопустимо – доработайте программу.

Лабораторная работа № 4 Перевод с помощью СУ-схемы

Пусть дана схема синтаксически управляемого перевода (теоретический материал раздела 4.2). Написать программу, которая будет выполнять перевод цепочек с одного языка на другой в соответствии с этой схемой. При невозможности выполнить перевод (цепочка не строится по правилам входной грамматики) необходимо выводить на экран соответствующее сообщение.

Правила СУ-схемы считывать из файла (предоставив пользователю возможность редактировать их на экране); цепочки вводить с клавиатуры, процесс перевода отображать на экране. Предусмотреть возможность выполнения перевода любого количества цепочек для заданной схемы.

На вход программы подаётся схема СУ-перевода (алфавиты входного и выходного языков, множество нетерминальных символов, правила вывода, целевой символ) и цепочка языка, которую необходимо перевести (может вводиться многократно, т.е. возможно перевести любое количество цепочек). Цепочка может быть и не относящейся к исходному языку…

Работа программы может быть организована по-разному. В качестве основного задания допустимо использовать материалы лабораторной работы №1. На первом этапе в соответствии с входной грамматикой схемы СУ-перевода попытаться сгенерировать цепочку, поданную на вход (поскольку её длина известна, можно генерировать только цепочки этой длины). При генерации фиксировать номера использованных правил. В случае успешной генерации выполнить второй этап – применяя номера использованных правил, получать одновременно входную и выходную цепочки по правилу вывода и соответствующему ему элементу перевода.

На выходе – отображение процесса перевода и результирующая цепочка языка.

Дополнительно: Вместо попыток генерации исходной цепочки сделать её разбор каким-либо способом из числа рассмотренных в теоретическом курсе. Например, использовать разбор с возвратами, нисходящий или восходящий (). В таком случае первый этап работы программы изменится на распознавание цепочки, т.е. на нём будет выполняться проверка цепочки построенным распознавателем. На выходе будут получены номера правил, по которым цепочка строилась. Второй этап – перевод с помощью схемы СУ-перевода – полностью совпадает с основным заданием.

***Контрольные вопросы:***

1. В чём отличие работы схемы СУ-перевода от порождения цепочек языка при помощи обычной грамматики?
2. Постройте схему СУ-перевода, которая будет переводить цепочки из алфавита {a,b,+,\*} из префиксной записи в постфиксную (в том же алфавите). Проверьте работу программы на этой схеме.

Лабораторная работа № 5   
 Перевод с помощью МП-преобразователя

Пусть дан преобразователь с магазинной памятью; написать программу, которая будет выполнять перевод цепочек с одного языка на другой с помощью заданного преобразователя (теоретический материал раздела 4.2). При невозможности выполнить перевод (цепочка не принадлежит исходному языку) необходимо выводить на экран соответствующее сообщение.

Исходный преобразователь вводить с клавиатуры в соответствии с определённым форматом. Ввод цепочек также производить с клавиатуры, выполняя его до тех пор, пока не возникнет желание закончить работу. Процесс перевода цепочки в виде последовательной смены конфигураций отображать на экране.

На вход программы подаётся ДМП-преобразователь (множество состояний, алфавиты входного и выходного языков, алфавит магазина, начальное состояние, начальное содержимое стека, множество заключительных состояний, функция переходов в виде списка правил) и проверяемая цепочка символов (аналогично лаб. раб. №3).

Рекомендуется за основу взять программу лабораторной работы №3, дополнив исходные данные выходным алфавитом, функцию переходов – в соответствии с определением преобразователя, а конфигурации – выходными цепочками.

Выход: отображение на экране процесса перевода цепочки в виде последовательной смены конфигураций преобразователя, результат перевода – полученная цепочка.

Проверить работу программы на примерах из лекций и задачах из контрольных вопросов.

Дополнительно:

Предоставить пользователю возможность не только вводить данные с клавиатуры, но и загружать преобразователь из файла (аналогично лабораторной работе №2).

***Контрольные вопросы:***

1. Как поведёт себя преобразователь, если поданная на его вход цепочка не может быть распознана (не принадлежит к заданному языку)?
2. Как соотносятся алфавиты исходного языка и того, на который выполняется перевод (должны совпадать, различаться…)? Поясните ответ.
3. Как поведёт себя Ваша программа при некорректном вводе? Например, функция переходов задана не в том формате, определена не для того количества параметров, использует алфавит, отличный от заданных…