Лексический анализ.

Лексический анализатор разбивает входной поток символов на лексемы и сопоставляет их токенам.
Каждый токен характеризуется набором правил, который определяет шаблон токена.
Например, шаблон токена «иденификатор» языка C++ можно определить следующим образом.
Идентификатор – это цепочка, составленная из латинских букв, цифр и знака _ . Цифра не может быть первым символом.

Существует два основных способа формального описания шаблонов. Первый, декларативный способ – это регулярные выражения. Второй, императивный способ – конечные автоматы.
Регулярное выражение записывается на специальном языке и определяет множество всех цепочек, составляющих токен.
Конечный автомат – это алгоритм, распознающий токен в заданной цепочке символов.
Оба способа взаимозаменяемы. Для любого регулярного выражения можно построить эквивалентный конечный автомат и наоборот. В реализации лексического анализатора, в любом случае, используется конечный автомат. Вопрос только в том, как он построен.
Существует множество утилит, которые по регулярному выражению строят автомат. Некоторые утилиты, такие как Lex, строят готовый программный код лексического анализатора, который затем встраивается в компилятор.

Мы не будем использовать регулярные выражения, а будем сразу описывать шаблоны токенов с помощью конечных автоматов. В ряде случаев построить конечный автомат гораздо проще, чем записать эквивалентное регулярное выражение.

Сущесвуют различные варианты задания конечного автомата. Мы будем использовать следующее формальное определение.

Конечный автомат – это упорядоченная пятерка параметров: M = {A, Q, T, q0,F}.

A – конечный алфавит входных символов
Q – конечное множество состояний
q0 – начальное состояние, принадлежащее Q
F – множество заключительных состояний, входящее или совпадающее с Q
T – отображение множества пар из QxA во множество Q

T называется функцией переходов конечного автомата и задается набором элементарных команд вида:

(qi,ak) -> qj

Комада означает, что автомат, находясь в состоянии qi, может прочитать входной символ ak и перейти в состояние qj.

Если функция переходов содержит хотя бы две команды (qi,ak) -> qj1 и (qi,ak) -> qj2, где qj1 отличается от qj2, то автомат называется недетерминированным(НКА), в противном случае автомат детерминированный(ДКА).

ДКА начинает работу в состоянии q0. считывая по одному символу входной цепочки a1a2 … am. Прочитанный символ переводит автомат в новое состояние в соответствии с функцией переходов. Если автомат, прочитав всю цепочку, оказывается в одном из заключительных состояний, то говорят, что он допускает эту цепочку, в противном случае – отвергает. Множесво всех допустимых цепочек формирует язык автомата.
Если атомат, находясь в состоянии qi, получает на входе символ ak, для которого нет подходящей команды перехода, он останавливается и сигнализирует об ошибке.

Доказана теорема о том, что для любого НКА можно построить эквивалентный ДКА, допускающий тот же язык.

Функцию переходов автомата можно наглядно представить в форме графической диаграммы.
При построении такой диаграммы будем использовать несколько соглашений.

Перенумеруем все состояния числами от 0 до n-1, где n общее число состояний. Числом 0 всегда будем обозначать начальное состояние.

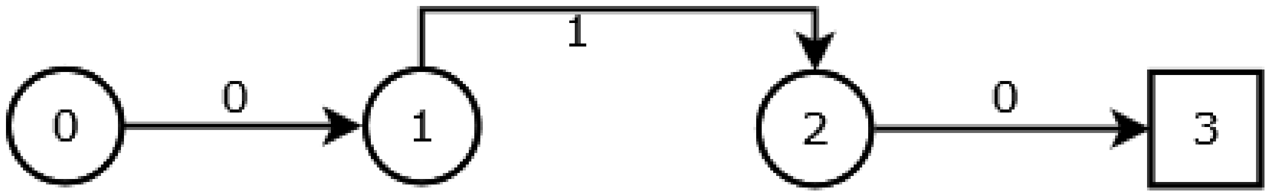
Обычное состояние будем изображать кружком, заключительное - квадратом, переход между состояниями ориентированной дугой со стрелкой на конце. Рядом с дугой запишем символ перехода.

Далее рассмотрим несколько примеров построения диаграмм для заданных языков.

Не существует какого-то общего алгоритма построения автомата по неформальному словесному описанию его языка, но есть очень полезный прием, основанный на методе матаматической индукции.

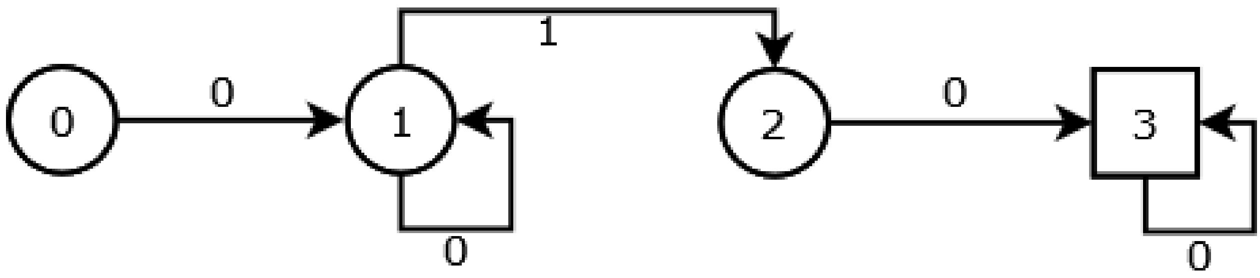
Алфавит: {0,1}

1. Цепочки, начинающиеся с непустой последовательности нулей, за ней следует одна единица, а за ней снова непустая последовательность нулей.
   Самая короткая цепочка, принадлежащая языку, – 010. Выберем эту цепочку в качестве базы индукции и построим диаграмму автомата, допускающего только ее.

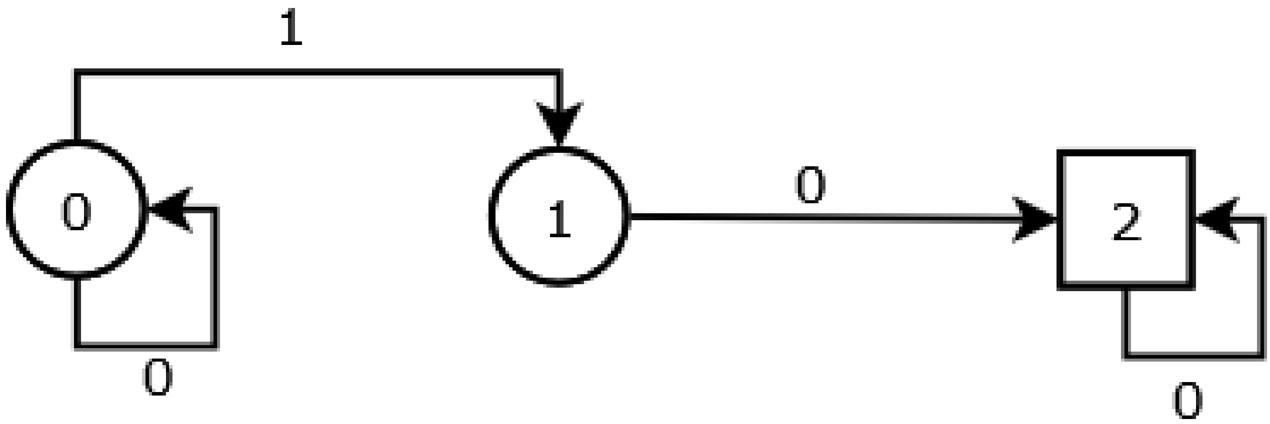


   Далее выполним два шага индукции. На первом шаге добавим петлю в состояние 1, на втором – в состояние 3. После каждого шага автомат будет допускать цепочки, более длинные, чем раньше.

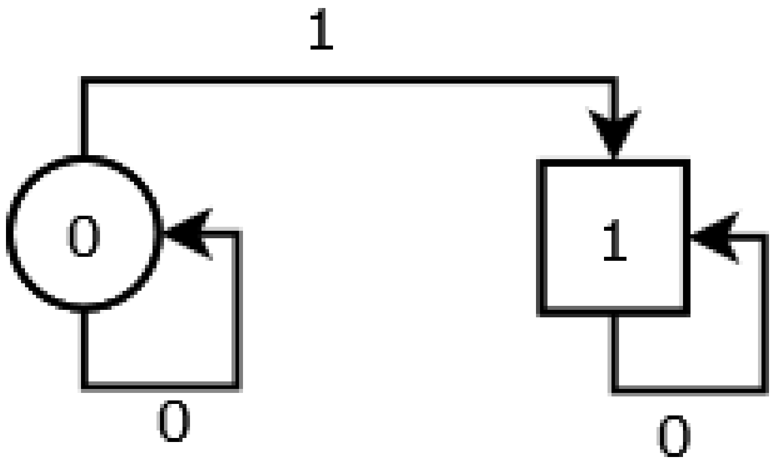
   В результата получим диаграмму автомата, допускающего заданный язык.



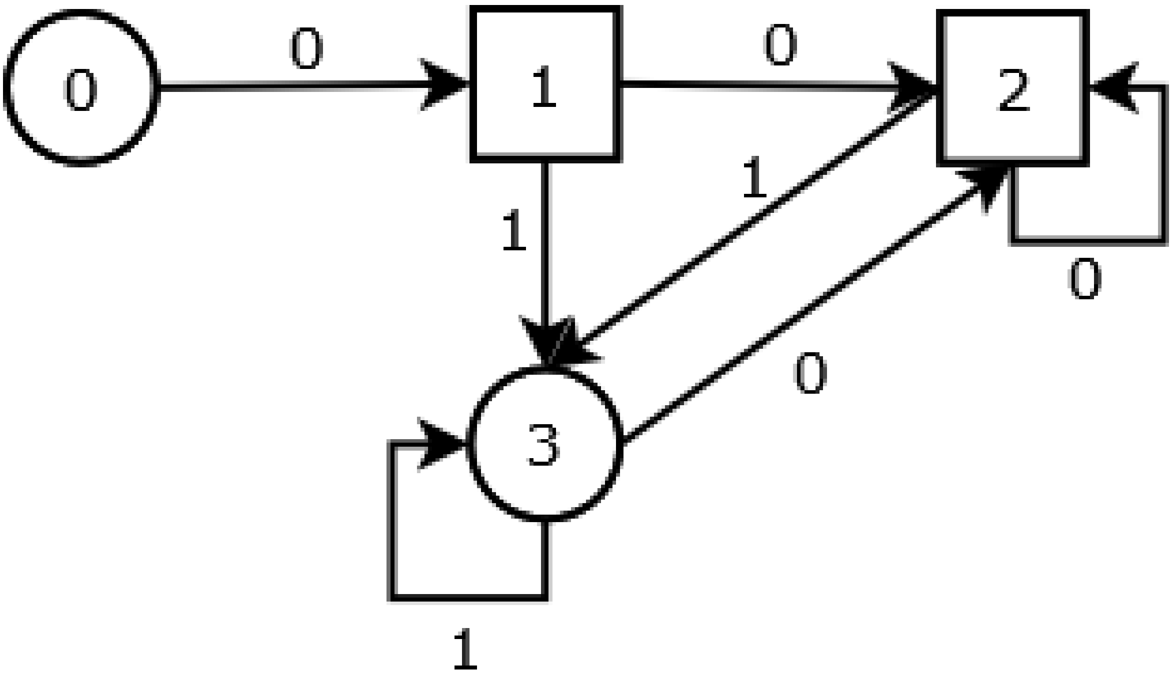
**2. Начальный префикс из нулей может быть пустым.
Самая короткая цепочка – 10.**



**3. Еще и конечный суффикс из нулей может быть пустым.
Самая короткая цепочка – 1**



**4. Цепочки, начинающиеся и оканчивающиеся нулем.
База индукции содержит две цепочки – 0 и 00**



NB!!! Чтобы автомат с конечным числом состояний допускал бесконечный язык, в его диашрамме должны присутствовать циклы.