**Регулярные выражения**

Регулярные выражения – мощное, гибкое и эффективное средство обработки текстов. Универсальные шаблоны регулярных выражений сами по себе напоминают миниатюрный язык программирования, предназначенный для описания и разбора текста. При дополнительной поддержке со стороны конкретной утилиты или языка программирования регулярные выражения способны вставлять, удалять, выделять и выполнять самые невероятные операции с текстовыми данными любого вида. Они бывают очень простыми, вроде команды поиска в текстовом редакторе, или очень сложными, как специализированные языки обработки текстов.

Регулярные выражения тесно связаны с теорией автоматов и формальных языков, поэтому они часто распознаются с использованием инструментов данных разделов математики.

Поэтому для начала разберемся с теорией автоматов.

**Конечные автоматы и регулярные языки**

Языком будем называть множество слов над некоторым алфавитом.

Определение

Детерминированным конечным автоматом будем называть пятерку (), где:

* – конечное множество состояний
* – конечное множество, называемое алфавитом
* – функция переходов
* – начальное состояние
* – множество конечных состояний

Определение

Автомат распознает (допускает) строку , если:

* существует последовательность такая что:

Языком распознаваемым автоматом называется .

Определение

Язык называется регулярным, если его распознает некоторый конечный автомат.

Различают детерминированные и недетерминированные конечные автоматы. Разница в том, что в детерминированных автоматах не может быть переходов по пустой строке, и множественных переходов по некоторому слову (то есть переходов в несколько состояний).

По сути функция перехода для недетерминированного автомата выглядит следующим образом:

, где множество всех подмножеств .

Для недетерминированного автомата в определении распознаваемого условие поменяется на .

Также различают недетерминированные конечные автоматы с -переходами.

Для таких автоматов функция перехода выглядит следующим образом:

, где множество всех подмножеств , где – пустая строка.

Также для таких автоматов меняется определение распознаваемых слов:

Определение

Конечный автомат с -переходами распознает (допускает) строку , если:

* существует последовательность такая что:

Языком распознаваемым автоматом называется .

Определение

-замыканием состояния конечного автомата будем называть множество

-замыканием есть множество вершин, достижимых из по ребрам с метками -замыканием множества состояний конечного автомата будем называть множество

В дальнейшем по НКА будем понимать НКА с переходами.

Известна следующая теорема:

Теорема

Для любого недетерминированного автомата существует эквивалентный ему детерминированный автомат (автоматы называются эквивалентными если языки, распознаваемые ими равны).

Регулярные языки замкнуты относительно следующих операций:

1. Объединение
2. Пересечение:
3. Дополнение:
4. Обращение:
5. Конкатенация:
6. Замыкание Клини:

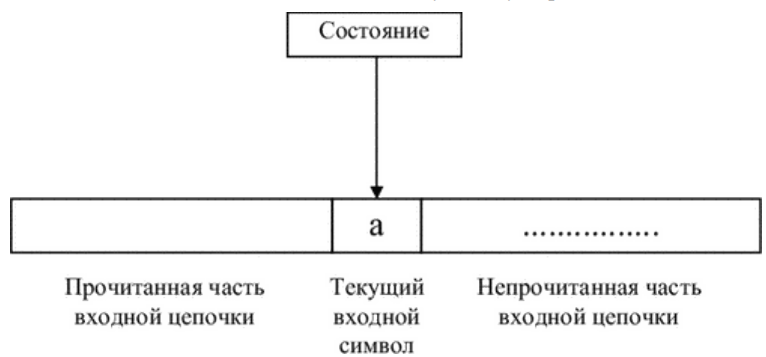
Соответственно регулярные выражения определяются рекурсивно:

1. регулярное выражение, обозначающее пустое множество
2. регулярное выражение, обозначающее множество
3. регулярное выражение, обозначающее множество
4. если и регулярные выражения, обозначающие регулярные языки и соответственно, то:
   1. минимальное надмножество множества , замкнутое относительно конкатенации.

язык, которому соответствует регулярное выражение

Регулярное выражение допускает строку , если .

Будем также использовать обозначение

Работа конечного автомата представляет собой некоторую последовательность шагов, или тактов. Такт определяется текущим состоянием управляющего устройства и входным символом, обозреваемым в данный момент входной головкой. Сам шаг состоит из изменения состояния и, возможно, сдвига входной головки на одну ячейку вправо:

Конфигурацией будем называть пару . начальная конфигурация, где – заключительная (допускающая). Таким образом после каждого такта наш автомат меняет конфигурацию. Более формально тактом определим следующим образом:

Определение

Тактом будем называть бинарное отношение на множестве конфигураций, такое что если , то .

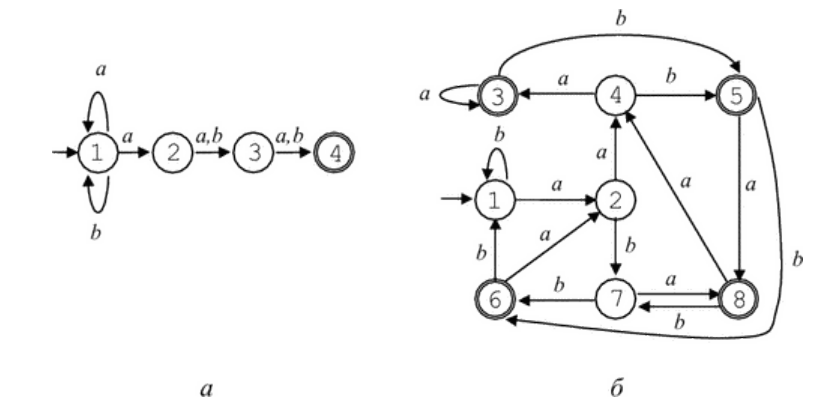
Будем также рассматривать соответствующие транзитивное и рефлексивно-транзитивное замыкание нашего бинарного отношения .

Таким образом автомат распознает строку тогда и только тогда, когда , где .

Конечные автоматы удобно представлять в виде ориентированного графа, в котором вершинами будут состояния нашего автомата, а между вершинами будет существовать ребро с меткой из алфавита , если соответствующий переход есть в функции переходов.

Пример

Рассмотрим регулярное выражение

Конечные автоматы, соответствующие данному регулярному выражению:

a – недетерминированный конечный автомат, б – детерминированный.

* Попробуем проанализировать цепочку с помощью автомата из пункта a.

Наш автомат может сделать следующую цепочку тактов:

.

Поскольку состояние 4 является заключительным, то наш автомат распознает строку .

* Попробуем проанализировать цепочку с помощью автомата из пункта б.

Поскольку автомат детерминированный то цепочка тактов определяется однозначно, и она равна:

Поскольку состояние 7 не является заключительным, то наш автомат не распознает строку .

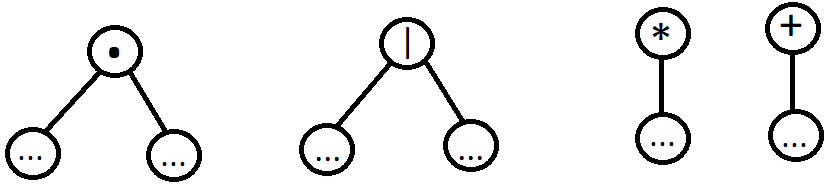
**Абстрактные синтаксические деревья**

Абстрактное синтаксическое дерево есть представление какого-либо вычисляемого выражения в виде дерева. Это может быть какое-нибудь арифметическое выражение, или же последовательность команд в языке программирования. В нашем случае дерево будет представлять регулярное выражение.

Внутренние вершины нашего дерево будут помечены некоторыми операндами нашего регулярного выражения. Варианты операндов:

* (“”) – конкатенация двух регулярных выражений.
* (“”) – дизъюнкция двух регулярных выражений.
* (“\*”) – замыкание Клини некоторого регулярного выражения
* (“+”) – замыкание Клини без пустых строк некоторого регулярного выражения.

Листьями будет вершина обозначающая некоторый символ алфавита (“sym”).

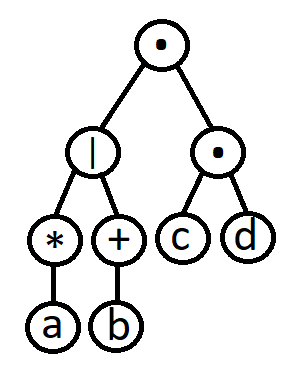


По сути абстрактное синтаксическое дерево для регулярного выражения задает в каком порядке можно проверять распознает ли регулярное выражение некоторую строку.

Нетрудно проанализировать распознавание некоторой строки регулярным выражением в случае если в синтаксическом дереве нету вершин (“”). Однако даже для простейшего случай конкатенации (например ) пришлось бы для строки находить множество префиксов, удовлетворяющих левой части конкатенации, множество суффиксов, удовлетворяющих правой части конкатенации и проверять что есть префикс и суффикс, составляющие всю строку, и распознаваемые соответствующими частями конкатенации. Это представляется довольно трудоемким и сложным процессом.

Поэтому использовать синтаксические деревья для распознавания регулярных выражений мы не будем. Вместо этого синтаксические деревья будут использоваться для построение конечных автоматов, с помощью которых уже можно легко проверить распознавание некоторой строки.

Пример

Построим абстрактное синтаксическое дерево для регулярного выражения .

**Алгоритмы**

Пускай у нас есть некоторое регулярное выражение . Для распознавания строк этим регулярным выражением нам необходимо построить для него конечный автомат (неважно детерминированный или недетерминированный).

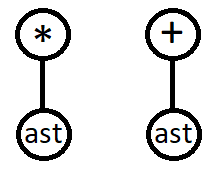
Однако прежде чем построить автомат нам необходимо построить абстрактное синтаксическое дерево.

**Алгоритм для построения абстрактного синтаксического дерева**

Для начала нам необходимо определиться с приоритетом операций. Наивысшим приоритетом у нас будут обладать операции , . Потом идет операция конкатенации, а последним идет операция дизъюнкции.

Алгоритм выглядит следующим образом:

* Храним последовательности АСД, которые будут объединены конкатенацией. Например для регулярного выражения мы имеем две последовательности: ,
* Идем по строке и смотрим текущий символ. Если символ равен:
  + “+” или “\*”, то мы берем последнее АСД из последней существующей последовательности АСД (назовем его ) и заменяем его на АСД следующего вида (в зависимости от случая):



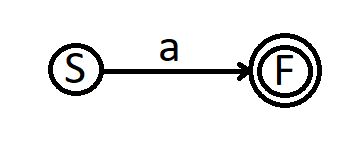
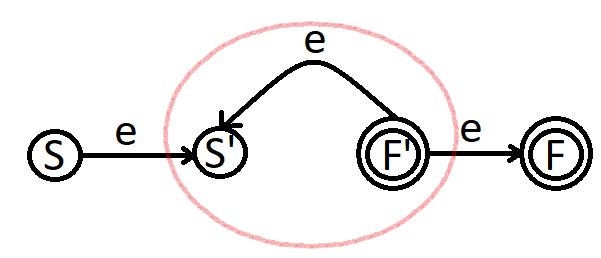
* + если мы встретили открывающую скобку, то мы рекурсивно строим АСД для выражения в скобках и добавляем его в последнюю последовательность.
  + если мы встретили закрывающую скобки, то мы объединяем последовательности дизъюнкцией и возвращаем полученное АСД из функции.
  + если мы встретили обычный символ, то добавляем в последнюю последовательность вершину с одним символом.
  + если мы встретили , то начинаем новую последовательность АСД и заполняем ее.
  + если закончилась строка, то мы объединяем последовательности дизъюнкцией и возвращаем полученное АСД из функции.

После отработки данного алгоритма мы получим АСД для нашего регулярного выражения.

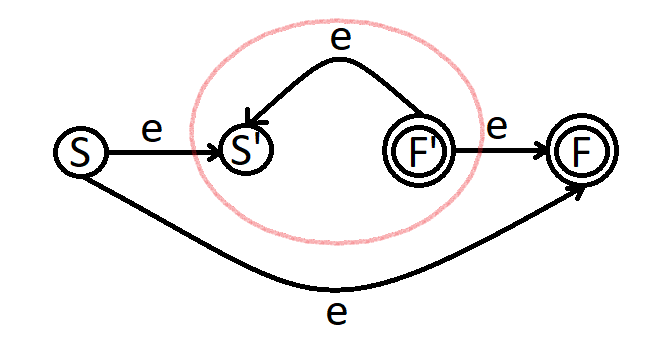
**Алгоритм для построения недетерминированного конечного автомата (НКА) из АСД**

У построенных нами автоматов будет лишь одна заключительная вершина, поэтому будем обозначать начальную и заключительную вершину и соответственно.

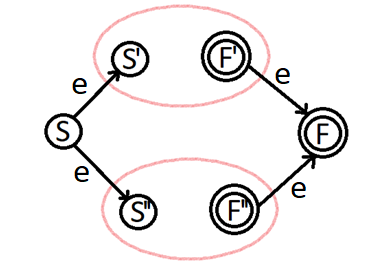
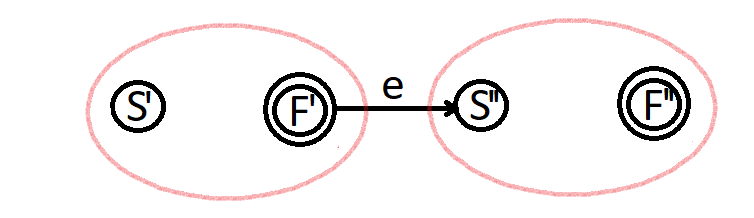
Проходимся по АСД и в зависимости от типа вершины делаем следующее:

* если у нас обычная символьная вершина, то строим автомат следующего вида:
* если у нас вершина типа (), то мы строим для внутреннего выражения автомат, а потом достраиваем его до автомата вида:

где и соответственно начальное и заключительное состояние для внутреннего регулярного выражения.

* если у нас вершина типа (), то мы строим для внутреннего выражения автомат, а потом достраиваем его до автомата вида:

где и соответственно начальное и заключительное состояние для внутреннего регулярного выражения.

* если у нас вершина типа дизъюнкции, то мы строим автоматы для левого и правого внутренних выражений, а потом достраиваем его до следующего вида:
* если у нас вершина типа конъюнкции, то мы строим автоматы для левого и правого внутренних выражений, а потом достраиваем его до следующего вида:

Определение

Переходом множества состояний () по некоторому символу будем называть множество:

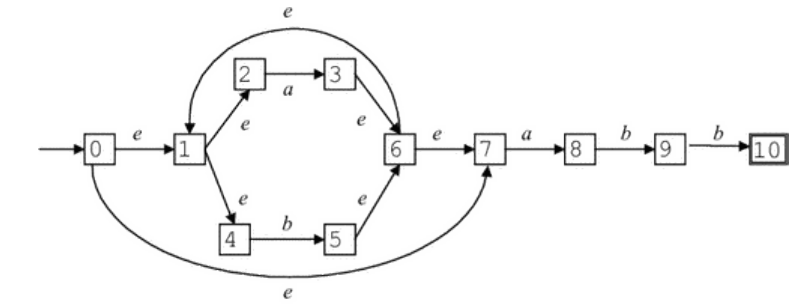
Переходом множества по некоторому символу есть множество вершин в которые можно перейти из некоторого состояния множества по ребру с меткой .

Алгоритм для распознавания строки:

* изначально инициируем множество
* далее для каждого символа делаем следующее:
* если после проделывания всех операций , то слово удовлетворяет регулярному выражению

Нетрудно заметить, что мы по сути по определению находим последовательность из определения распознаваемости строки конечным автоматом с -переходами.

Пример

Рассмотрим следующим НКА:

Проверим на нем строку

* Изначально
* Далее делаем итерации:
* Поскольку , то строка распознается нашим регулярным выражением.

**Алгоритм для построения недетерминированного конечного автомата (НКА) из АСД**