

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

ПРИМЕНЕНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ ДЛЯ
РАСПОЗНАВАНИЯ ПОДСТРОК В ТЕКСТЕ

Курсовой проект

Казакова Александра Валерьевича
студента 3 курса,
специальности «информатика»

Научный руководитель:
Лукьянов Иван Денисович
ассистент кафедры
ДМА

Минск, 2020

Оглавление

ВВЕДЕНИЕ	3
КОНЕЧНЫЕ АВТОМАТЫ И РЕГУЛЯРНЫЕ ЯЗЫКИ	4
АБСТРАКТНЫЕ СИНТАКСИЧЕСКИЕ ДЕРЕВЬЯ.....	8
АЛГОРИТМЫ	10
Алгоритм для построения абстрактного синтаксического дерева.....	10
Алгоритм для построения недетерминированного конечного автомата (НКА) из АСД ...	11
Алгоритм для построения детерминированного конечного автомата (ДКА) из НКА	14
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ	17
ЗАКЛЮЧЕНИЕ	19

ВВЕДЕНИЕ

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 используются в таких сервисах как:

- Google Analytics и Яндекс.Метрика: для работы с фильтрами и отчетами, настройка целей, сегментов и аудиторий.
- Google Tag Manager: для настройки триггеров и переменных.
- Wordstat: для подбора семантики.
- Key Collector: для подбора семантики и фильтрации данных.
- Excel и Google Таблицы: для фильтрации и извлечения нужных данных.

Многие современные языки программирования имеют встроенную поддержку регулярных выражений. Среди них ActionScript, Perl, Java, PHP, JavaScript, языки платформы .NET Framework, Python, Tcl, Ruby, Lua, Gambas, C++ (стандарт 2011 года), Delphi, D, Нахе и другие.

Регулярные выражения тесно связаны с теорией автоматов и формальных языков, поэтому они часто распознаются с использованием инструментов данных разделов математики. Поэтому для написания собственной программы распознавания регулярных выражений придется разобраться с основными аспектами теории конечных автоматов, понять какие возможности она дает и реализовать соответствующие алгоритмы.

Соответственно целью данного курсового проекта является написание инструмента для распознавания регулярного выражения в тексте.

КОНЕЧНЫЕ АВТОМАТЫ И РЕГУЛЯРНЫЕ ЯЗЫКИ

Языком будем называть множество слов над некоторым алфавитом.

Определение

Детерминированным конечным автоматом будем называть пятерку $(Q, \Sigma, \delta, q_0, F)$, где:

- Q – конечное множество состояний
- Σ – конечное множество, называемое алфавитом
- $\delta : Q \times \Sigma \rightarrow Q$ – функция переходов
- $q_0 \in Q$ – начальное состояние
- $F \subseteq Q$ – множество конечных состояний

Определение

Автомат $A = (Q, \Sigma, \delta, q_0, F)$ распознает (допускает) строку $w = \overline{w_1 \dots w_n}$, если:

- $w_i \in \Sigma, \forall i = \overline{1, n}$
- существует последовательность $q_0 = r_0, r_1, \dots, r_n$ такая что:
 - $r_n \in F$
 - $r_{i+1} = \delta(r_i, w_i), \forall i = \overline{0, n-1}$

Языком распознаваемым автоматом A называется $L = \{w \mid A \text{ допускает } w\}$.

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:

$\delta : Q \times \Sigma \rightarrow B(Q)$, где $B(Q)$ – множество всех подмножеств Q .

Для недетерминированного автомата в определении распознаваемого условия $r_{i+1} = \delta(r_i, w_i)$ поменяется на $r_{i+1} \in \delta(r_i, w_i)$.

Также различают недетерминированные конечные автоматы с ε -переходами.

Для таких автоматов функция перехода выглядит следующим образом:

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow B(Q)$, где $B(Q)$ – множество всех подмножеств Q , где ε – пустая строка.

Также для таких автоматов меняется определение распознаваемых слов:

Определение

Конечный автомат с ε -переходами $A = (Q, \Sigma, \delta, q_0, F)$ распознает (допускает) строку $w = \overline{w_1 \dots w_n}$, если:

- $w_i \in \Sigma, \forall i = \overline{1, n}$
- существует последовательность r_0, r_1, \dots, r_n такая что:
 - $r_0 \in E(q_0)$
 - $r_n \in F$
 - $r_{i+1} \in E(r')$, где $r' \in \delta(r_i, w_i), \forall i = \overline{0, n-1}$

Языком распознаваемым автоматом A называется $L = \{w \mid A \text{ допускает } w\}$.

Определение

ε -замыканием состояния конечного автомата будем называть множество $E(q) = \{q' \in Q \mid \exists q = q_0, q_1, \dots, q_k = q', q_{i+1} \in \delta(q_i, \varepsilon), i = \overline{0, n-1}\}$

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

$$E(R) = \bigcup_{q \in R} E(q)$$

В дальнейшем под НКА будем понимать НКА с ε 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: $A \cup B = \{w \mid w \in A \text{ или } w \in B\}$
2. Пересечение: $A \cap B = \{w \mid w \in A \text{ и } w \in B\}$
3. Дополнение: $\bar{A} = \{w \in \Sigma \cup \{\varepsilon\} \mid w \notin A\}$
4. Обращение: $A^R = \{a_1 \dots a_k \mid a_k \dots a_1 \in A\}$
5. Конкатенация: $A \circ B = \{vw \mid v \in A \text{ и } w \in B\}$
6. Замыкание Клини: $A^* = \{w_1 \dots w_k \mid k \geq 0 \text{ и } w_i \in A \text{ для всех } i\}$

Соответственно регулярные выражения определяются рекурсивно:

1. \emptyset — регулярное выражение, обозначающее пустое множество
2. e — регулярное выражение, обозначающее множество $\{e\}$
3. a — регулярное выражение, обозначающее множество $\{a\}$

4. если p и q регулярные выражения, обозначающие регулярные языки P и Q соответственно, то:

a. $pq = \{uw \mid u \in P \text{ и } w \in Q\}$

b. $p|q = \{u \mid u \in P \text{ или } u \in Q\}$

c. p^* — минимальное надмножество множества P , замкнутое относительно конкатенации.

$L(R)$ — язык, которому соответствует регулярное выражение R

Регулярное выражение допускает строку w , если $w \in L(r)$.

Будем также использовать обозначение $p^+ = pp^*$

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:



Рисунок 1.

Конфигурацией будем называть пару $(q, w) \in Q \times T^* (T^* = T \cup \{\varepsilon\})$. (q_0, w) — начальная конфигурация, (q, ε) , где $q \in F$ — заключительная (допускающая). Таким образом после каждого такта наш автомат меняет конфигурацию. Более формально тактом определим следующим образом:

Определение

Тактом будем называть бинарное отношение \vdash на множестве конфигураций, такое что если $q' \in \delta(q, a)$, то $(q, aw) \vdash (q', w), \forall w \in T^*$.

Будем также рассматривать соответствующие транзитивное и рефлексивно-транзитивное замыкание нашего бинарного отношения \vdash^+, \vdash^* .

Таким образом автомат распознает строку w тогда и только тогда, когда $(q_0, w) \vdash^*(q, \varepsilon)$, где $q \in F$.

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 $\Sigma \cup \{\varepsilon\}$, если соответствующий переход есть в функции переходов.

Пример

Рассмотрим регулярное выражение $R = (a|b) * a(a|b)(a|b)$

Конечные автоматы, соответствующие данному регулярному выражению:

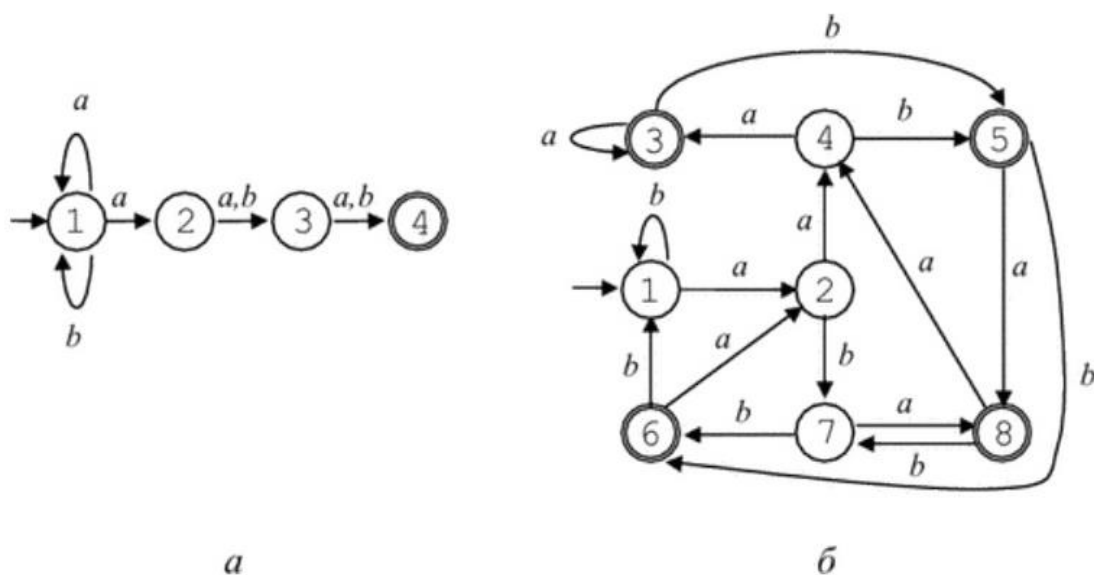


Рисунок 2

а – недетерминированный конечный автомат, б – детерминированный.

- Попробуем проанализировать цепочку $w = ababa$ с помощью автомата из пункта а.

Наш автомат может сделать следующую цепочку тактов:

$$(1, ababa) \vdash (1, baba) \vdash (1, aba) \vdash (2, ba) \vdash (3, a) \vdash (4, e).$$

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

- Попробуем проанализировать цепочку $w = ababab$ с помощью автомата из пункта б.

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

$$(1, ababab) \vdash (2, babab) \vdash (7, abab) \vdash (8, bab) \vdash (7, ab) \vdash (8, b) \vdash (7, e)$$

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

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 дерева 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”).

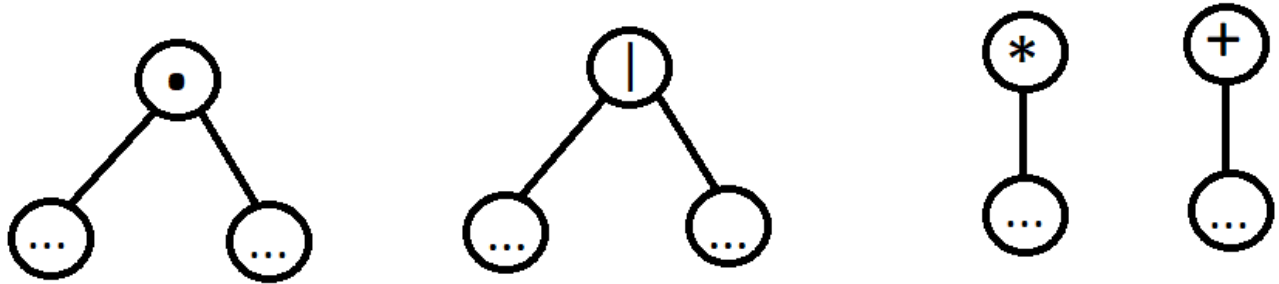


Рисунок 3

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 a^+b^+) 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 построения конечных автоматов, с помощью которых уже можно легко проверить распознавание некоторой строки.

Пример

Построим абстрактное синтаксическое дерево для регулярного выражения $R = (a^*|b^+)(cd)$.

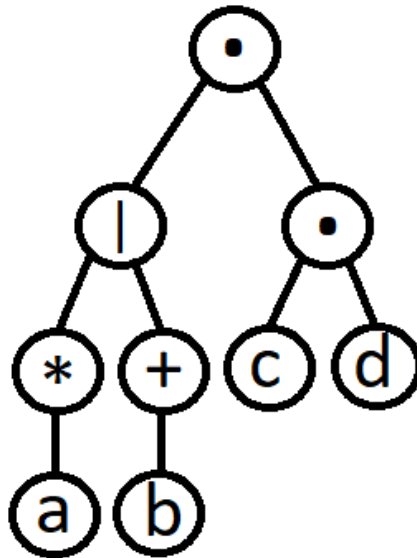


Рисунок 4

АЛГОРИТМЫ

Пусть у нас есть некоторое регулярное выражение R . 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 $R = (a|b)cd^*|f(g^*e^+)^+$ мы имеем две последовательности: $s_1 = \{\text{АСД}((a|b)), \text{АСД}(c), \text{АСД}(d^*)\}$, $s_2 = \{\text{АСД}(f), \text{АСД}((g^*e^+)^+)\}$
- Идем по строке и смотрим текущий символ. Если символ равен:
 - " $+$ " или " $*$ ", то мы берем последнее АСД из последней существующей последовательности АСД (назовем его ast) и заменяем его на АСД следующего вида (в зависимости от случая):

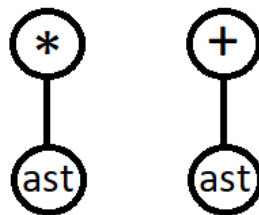


Рисунок 5

- 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 S и F соответственно.

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

- если у нас обычная символьная вершина, то строим автомат следующего вида:

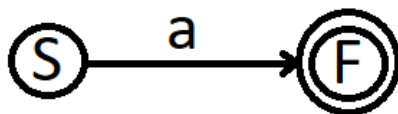


Рисунок 6

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

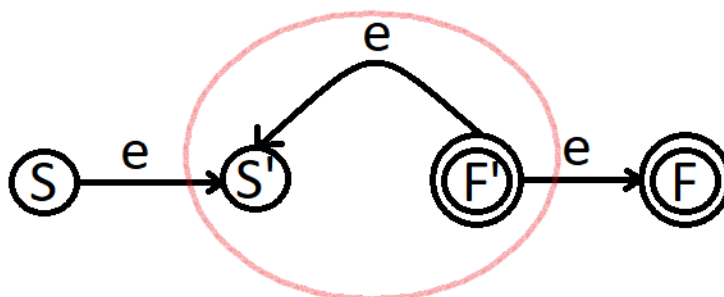


Рисунок 7

где S' и F' 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:

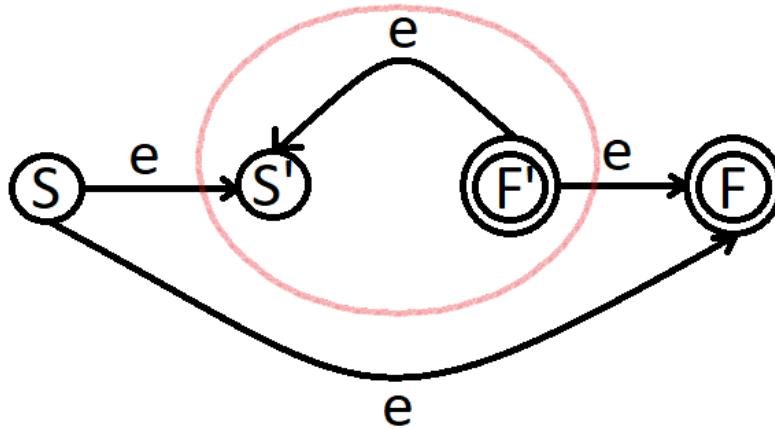


Рисунок 8

где S' и F' 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:

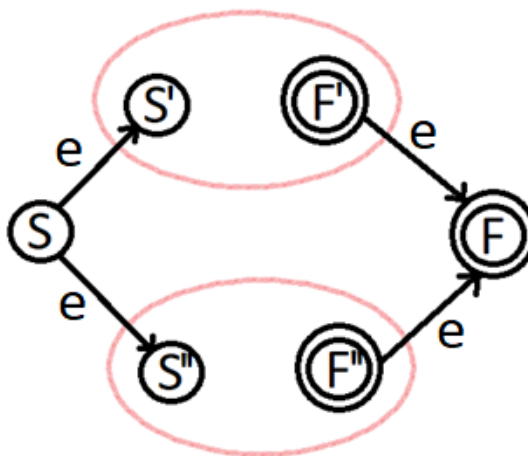


Рисунок 9

- 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 (R) по некоторому символу (a) будем называть множество:

$$Move(R, a) = \bigcup_{q \in R} \{q' \in Q \mid q' \in \delta(q, a)\}$$

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

Алгоритм для распознавания строки:

- изначально иницилируем множество $R = N_e(\{q_0\})$
- далее для каждого символа $w_i \in w$ делаем следующее:

$$R := Move(R, w_i)$$

$$R := E(R)$$

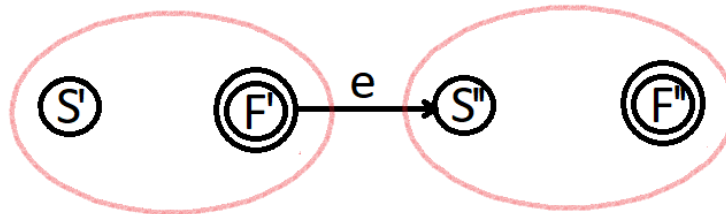


Рисунок 10

- если после проделывания всех операций $R \cap F \neq \emptyset$, то слово удовлетворяет регулярному выражению

Нетрудно заметить, что мы по сути по определению находим последовательность r_0, \dots, r_n из определения распознаваемости строки конечным автоматом с ε -переходами.

Пример

Рассмотрим следующий НКА:

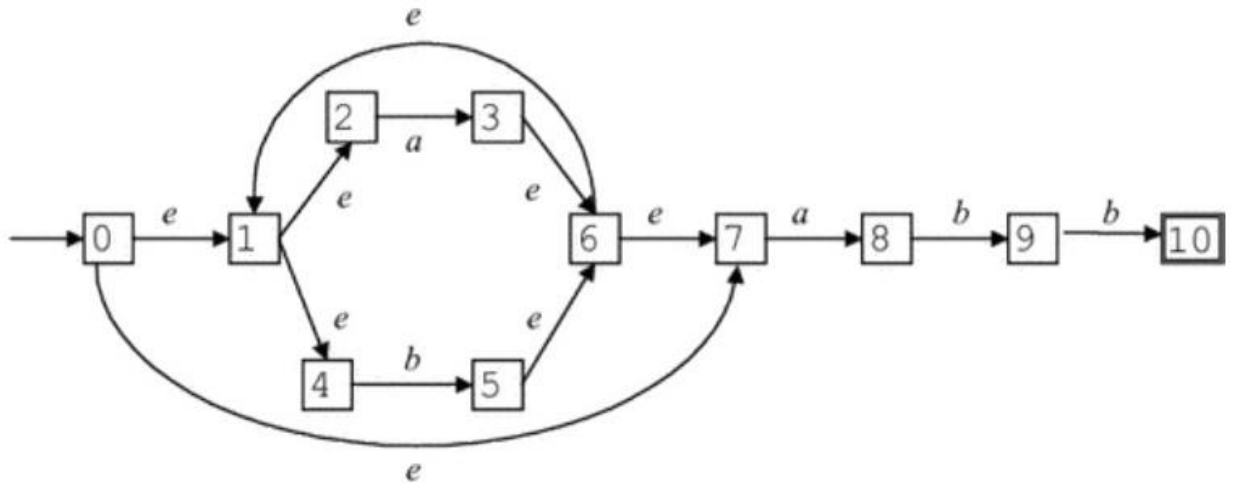


Рисунок 11

Проверим на нем строку $w = aabb$

- Изначально $R = \{0, 1, 2, 4\}$
- Далее делаем итерации:
 - $R := \{3, 6, 7, 1, 2, 4\}$
 - $R := \{8, 3, 6, 7, 1, 2, 4\}$
 - $R := \{9, 5, 6, 7, 1, 2, 4\}$
 - $R := \{10, 5, 6, 7, 1, 2, 4\}$
- Поскольку $R \cap F = \{10\} \neq \emptyset$, то строка распознается нашим регулярным выражением.

Алгоритм для построения детерминированного конечного автомата (ДКА) из НКА

Нам необходимо построить новый детерминированный конечный автомат $\bar{A} = (\bar{Q}, \Sigma, \bar{\delta}, \bar{q}_0, \bar{F})$ (алфавит останется таким же) из НКА A .

В новом автомате состояниями будут неупорядоченные множества состояний автомата A , поскольку на самом-то деле в автомате \bar{A} состояние будет включать в себя состояния автомата A .

Изначально $\bar{Q}, \bar{\delta}$.

- $\bar{q}_0 = E(q_0)$
- Добавляем \bar{q}_0 в \bar{Q} , как не посещенное состояние.

- Пока в \bar{Q} есть не посещенное состояние делаем следующее:
 - Берем не посещенное состояние R .
 - Помечаем его как посещенное
 - Для каждого входного символа a из алфавита Σ рассматриваем $S = E(\text{Move}(R, a))$
 - Если S не пустое множество
 - Если S не содержится в \bar{Q} , то добавляем S в \bar{Q} как не помеченное состояние
 - Определяем $\bar{\delta}(R, a) = S$
- Определяем $\bar{F} = \{R | R \in \bar{Q}, R \cap F \neq \emptyset\}$

Если объяснять на пальцах, то алгоритм по сути перебирает все возможные наборы состояний, которые могут быть посещены некоторой последовательностью символов из алфавита. Все эти наборы являются новыми состояниями для детерминированного автомата. Из любого набора состояний можно единственным образом перейти в некоторый другой набор состояний. Эти переходы и записываются в функцию переходов нового конечного детерминированного автомата. Те состояния, которые включают в себе хотя бы одно заключительное состояние исходного НКА и будут являться заключительными состояниями для нового ДКА.

Пример

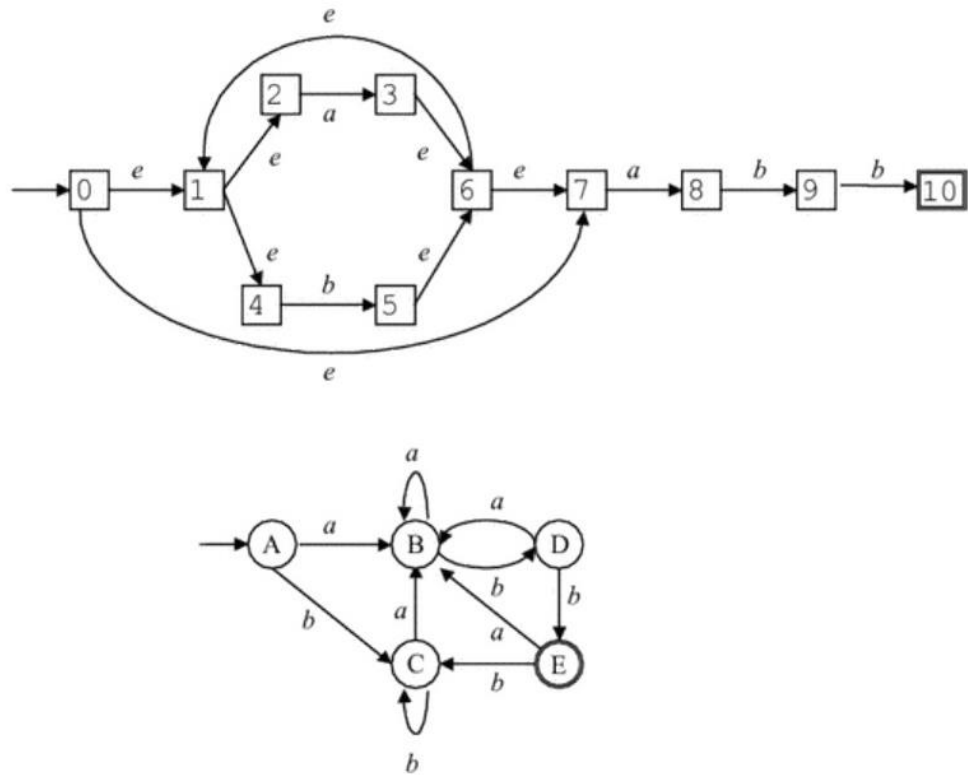


Рисунок 12

Снизу изображен результат работы алгоритма на НКА сверху.

Преимущества ДКА над НКА:

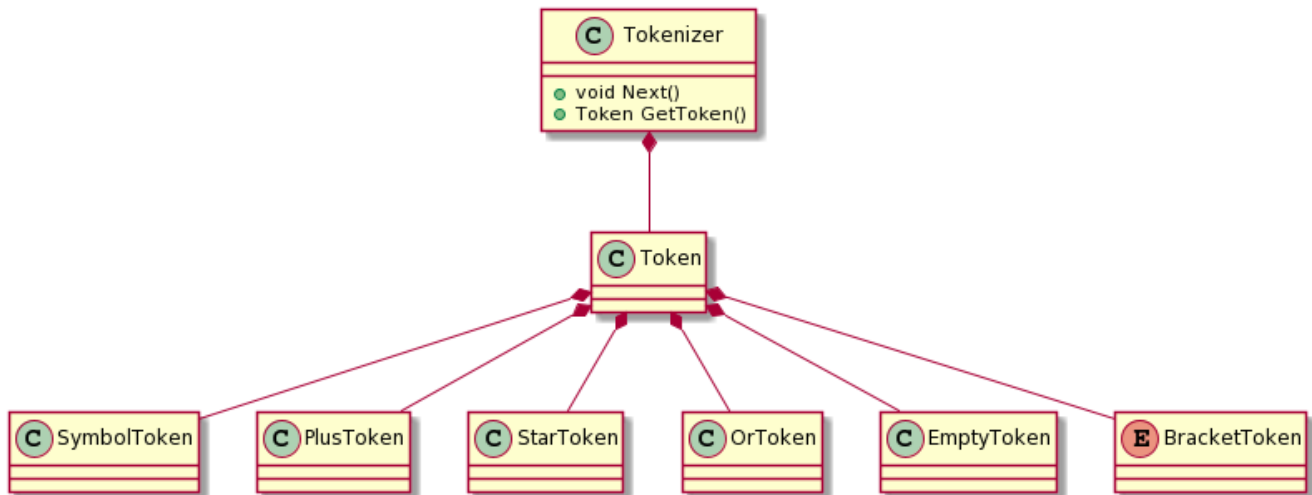
- в ДКА нету ε -переходов, а значит при проверки строки не надо делать затратную операцию $E(R)$
- Также в ДКА в любой момент времени в R рассматривается лишь одно состояние, что значительно сокращает время переходов.

Соответственно с помощью ДКА можно намного быстрее обработать строку.

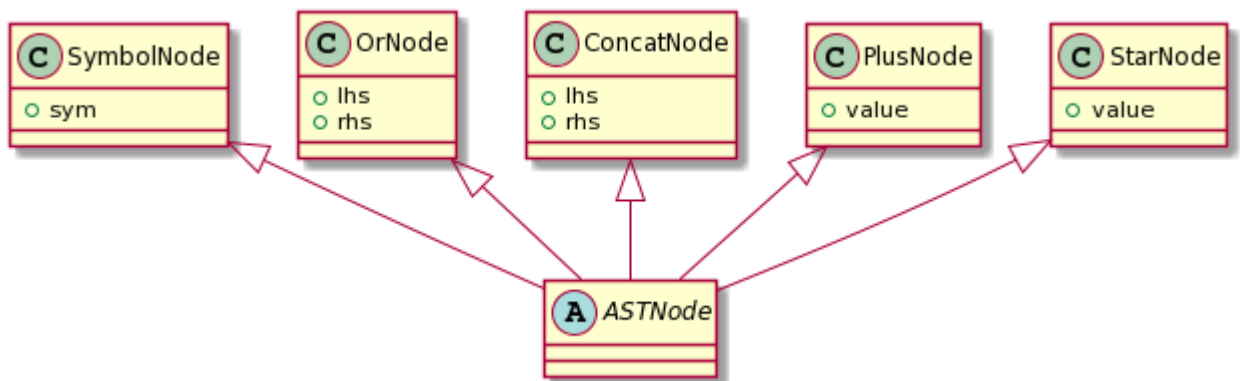
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Сделаны классы из алгоритмов.

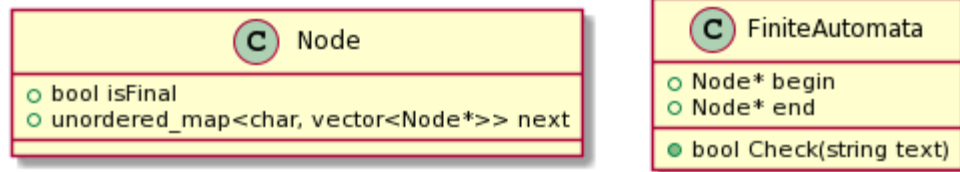
Для начала есть класс-разборщик элементов регулярного выражения (Tokenizer). Он разбивает строку-образец на элементы и позволяет получить текущий элемент или взять следующий. Конструируется этот класс из потока данных с нашим регулярным выражением.



После реализован класс абстрактного синтаксического дерева (если быть точным, то вершины абстрактного синтаксического дерева). Также есть методы, строящие АСД из потока данных содержащего регулярное выражение.



Также есть класс реализующий конечные автоматы. Внутри он содержит определение класса вершины.



Также есть соответствующие методы, которые позволяют из абстрактного синтаксического дерева построить недетерминированный конечный автомат и потом из недетерминированного конечного автомата построить детерминированный конечный автомат. НКА и ДКА отличаются лишь тем что ДКА это НКА в котором нету переходов по нулевому символу и множество состояний после перехода по символу у ДКА это массив из одного символа (т.е. НКА и ДКА реализованы одним классом).

Весь код, вместе с тестами и примерами использования можно найти по ссылке: <https://github.com/Karial/regex-cpp-parser>.

ЗАКЛЮЧЕНИЕ

Таким образом в данной работе были исследованы и реализованы алгоритмы для распознавания строк, соответствующих некоторому регулярному выражению. Однако построенная при этом утилита все еще является неполноценной и распознает лишь небольшой спектр регулярных выражений.

Задача нахождения всех подстрок, удовлетворяющих некоторому регулярному выражению пока что решается лишь перебором всех подстрок текста и распознавание каждой из них регулярным выражением. Однако хотелось бы выполнять данную задачу быстрее, поскольку уже на небольших текстах это будет долго (поскольку количество операций получается $O(|\text{длина текста}|^3)$).

Не были реализованы следующие возможности регулярных выражений:

- Группы символов. Например $\backslash d$ – цифровой символ, $\backslash S$ – непробельный символ, $\backslash w$ – буквенный или цифровой символ или знак подчёркивания.
- Интервалы символов $[e - h]$ – символы с “e” по “h”, $[3 - 7]$ – цифры с 3 по 7.
- Определенное число символов $a\{2, 4\}$ – от 2 до 4 символов “a”.
- Обратные ссылки

Данные дополнения к функциональности регулярных выражений не изменяют общего подхода для их распознавания, а лишь порождает рассмотрение большего числа случаев и добавление нового функционала в конечные автоматы (например, по дуге можно будет перейти не по одному символу, а по нескольким).