Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа искусственного интеллекта

Направление 02.03.01 «Математика и компьютерные науки»

Дисциплина «Математическая логика и теория автоматов»

Отчёт по лабораторной работе №2

«Построение автомата-распознавателя»
Вариант 19

Студент: Башарина Е.А., гр. 3530201/90101

Преподаватель: Востров А.В.

Санкт-Петербург — 2022

Содержание

Введение

Регулярные выражения - это язык для поиска цепочек символов, удовлетворяющих заданным условиям. Их применения включают в себя широкий спектр задач, таких как проверка входной строки на корректность, извлечение из текста определённой подстроки и сравнение строки с заданным шаблоном. Множество цепочек, построенных по определенным правилам (цепочек, удовлетворяющих регулярному выражению), является автоматным языком. Таким образом, распознавание таких цепочек символов может быть реализовано посредством конечного автомата-распознавателя. В данном отчете содержится описание выполнения лабораторной работы, в ходе которой был построен и программно реализован конечный автомат-распознаватель по заданному регулярному выражению. Кроме того, в рамках лабораторной работы реализована генерации случайной корректной строки.

Данный отчёт содержит детерменированный и недетерменированный конечные автоматы, построенные по регулярному выражению, соответстующему варианту задания. Кроме того, представлена программная реализация построенного автомата на языке Python.

Цель работы: по заданному регулярному выражению построить вначале недетерминированный конечный автомат, затем детерминировать его. Реализовать программу, которая проверяет введенный текст, через реализацию конечного автомата. Также необходимо реализовать функцию случайной генерации верной строки по полученному конечному автомату.

Допускается наличие переходов в конечном автомате, заданных диапазонами символов. От реализации программы ожидаются три возможных реакции:

1. строка соответствует регулярному выражению;
2. строка не соответствует регулярному выражению;
3. в строке содержатся символы, не относящиеся к алфавиту.

Регулярное выражение, указанное в задании: отображение времени 2.

```
^([01]\d|2[0-3]):[0-5]\d\s[ap]m$
```

1 Математическое описание

1.1 Конечный автомат-распознаватель

Конечный автомат-распознаватель $A =< S, X, s_0, \delta, F >$, где:

- S — конечное непустое множество состояний;
- X — конечное непустое множество входных сигналов (иначе говоря, входной алфавит);
- $s_0 \in S$ — множество начальных состояний;
- $\delta : S \times X \to S$ — функция переходов;
- $F \in S$ — множество финальных состояний.

Конечный автомат-распознаватель $A =< S, X, s_0, \delta, F >$ допускает входную цепочку $\alpha \in X^*$, если α переводит его из начального состояния в одно из конечных. Множество всех цепочек, допускаемых автоматом A, образует язык, допускаемый A.

Различают детерминированные КА — автоматы, в которых следующее состояние однозначно определяется текущим состоянием и выход зависит только от текущего состояния и текущего входа, и недетерминированные КА, следующее состояние у которых в общем случае неопределённо и, соответственно, не определён выходной сигнал.

1.2 Построенный конечный автомат

По заданному регулярному выражению был построен недетерминированный конечный автомат (рис. 1). Недетерминированный конечный автомат задаётся следующим образом:

1. S = {s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11}
2. X = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, p, m, :, ' '}
3. s_0 = s1
4. δ = {
   δ(s1, [0, 1]) = {s2},
   δ(s1, 2) = {s4},
   δ(s2, [0-9]) = {s3},
   δ(s3, :) = {s6},
   δ(s4, [0-3]) = {s5},
   δ(s6, [0-5]) = {s7},
   δ(s7, [0-9]) = {s8},
   δ(s8, ' ') = {s9},
   δ(s9, [a, p]) = {s10},
   δ(s10, m) = {s11}
   }
5. F = s11

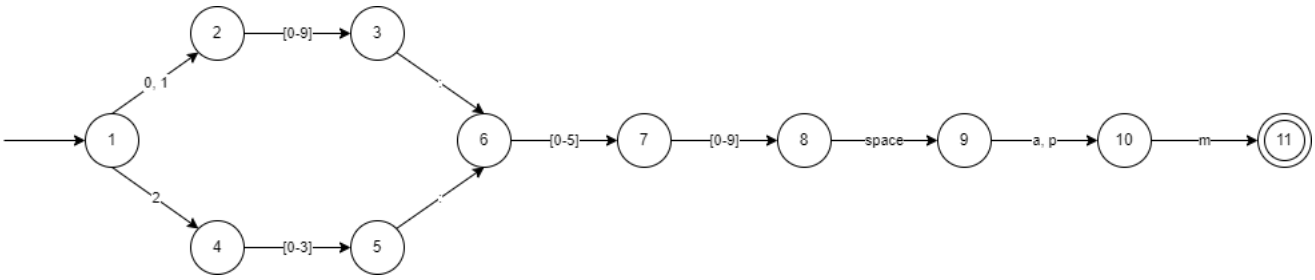

Рис. 1: Недетерминированный конечный автомат

Впоследствии построенный автомат был детерминирован (рис. 2). Детерминированный конечный автомат задаётся следующим образом:

1. S = {s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, mistake}
2. X = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, p, m, :, ' '}
3. s_0 = s1
4. δ = {
   δ(s1, [0, 1]) = {s2},
   δ(s1, 2) = {s4},
   δ(s2, [0-9]) = {s3},
   δ(s3, :) = {s6},
   δ(s4, [0-3]) = {s5},
   δ(s6, [0-5]) = {s7},
   δ(s7, [0-9]) = {s8},
   δ(s8, ' ') = {s9},
   δ(s9, [a, p]) = {s10},
   δ(s10, m) = {s11},
   δ(s1, [0, 1, 3-9, , a, p, m, :]) = {mistake},
   δ(s2, [, a, p, m, :]) = {mistake},
   δ(s3, [0-9, , a, p, m]) = {mistake},
   δ(s4, [4-9, , a, p, m, :]) = {mistake},
   δ(s5, [0-9, , a, p, m]) = {mistake},
   δ(s6, [6-9, , a, p, m, :]) = {mistake},
   δ(s7, [, a, p, m, :]) = {mistake},
   δ(s8, [0-9, a, p, m, :]) = {mistake},
   δ(s9, [0-9, , m, :]) = {mistake},
   δ(s10, [0-9, , a, p, :]) = {mistake},
   δ(s11, [0-9, , a, p, m, :]) = {mistake},
   δ(mistake, [0-9, , a, p, m, :]) = {mistake},
   }
5. F = s11

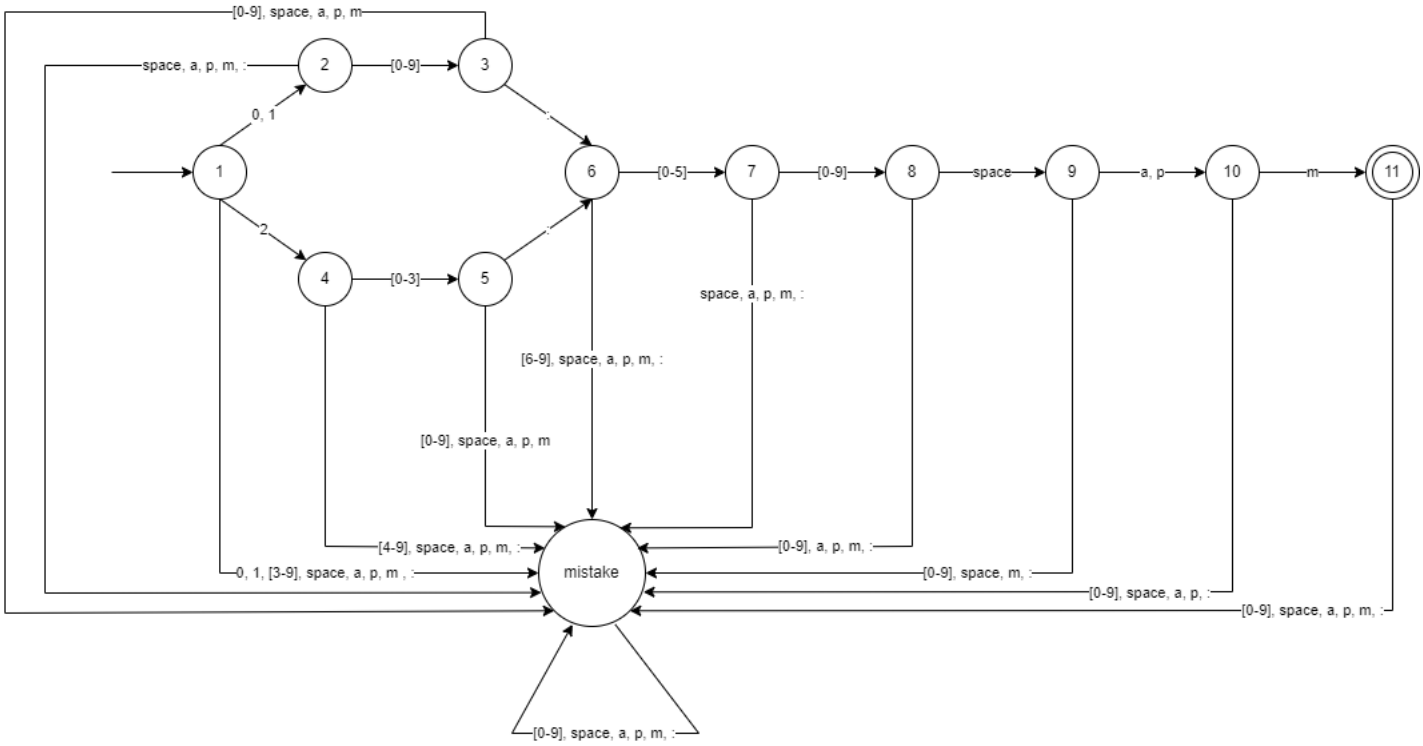

Рис. 2: Детерминированный конечный автомат

1.3 Регулярные множества и выражения

Регулярные множества — это множества цепочек, построенных из символов конечного словаря по определенным правилам.

Регулярные множества, как множества цепочек, построенные над конечным словарем по определенным правилам — это **регулярные языки**[1].

Регулярное множество в алфавите Σ определяется рекурсивно следующим образом:

- $\emptyset$ — регулярное множество в алфавите Σ;
- ε — регулярное множество в алфавите Σ (ε — пустая цепочка);
- a —регулярное множество в алфавите Σ для каждого $a \in \Sigma$.

Если P и Q регулярные множества, то регулярны следующие **операции** над ними:

- $P \cup Q$ — объединение;
- PQ — конкатенация, то есть множество $pq, p \in P, q \in Q$;
- P^* — итерация ($P^* = \bigcup_{n=0}^{\infty} P^n$).

Регулярные множества определяют **семантику** регулярных выражений. Регулярные выражения представляют конечный порождающий механизм для регулярного языка[1].

Если $R_1 R_2$ — **регулярные выражения**, то регулярными выражениями будут:

- $R_1 + R_2$ — их сумма;
- $R_1 R_2$ — их конкатенация;
- R_1^* — итерация и усеченная итерация R_1^+.

В соответствии с **теоремой Клини** любой язык, распознаваемый конечным автоматом, может быть задан регулярным выражением, и для любого языка, заданного регулярным выражением, может быть построен конечный автомат, его распознающий. Иными словами, классы регулярных множеств и автоматных языков совпадают.

1.4 Языки и грамматики

Язык (над конечным словарем V) — это произвольное множество конечных цепочек над этим словарем — L_v.

Грамматика — это совокупность грамматических правил некоторого языка. Представляет из себя теоретическую модель языка, состоящую из отдельных гипотез структурной организации языка.

Порождающая грамматика языка L — это конечный набор правил, позволяющих строить все предложения, принадлежащие языку L, и применение которых не дает ни одного предложения, не принадлежащего L[2].

С помощью распознающей грамматики задается критерий принадлежности произвольной цепочки данному языку. Фактически, **распознающая грамматика** — это некоторый алгоритм, принимающий произвольную цепочку символов над словарем V и дающий на выходе один из двух возможных ответов:

- цепочка *принадлежит* языку L;
- цепочка *не принадлежит* языку L.

Кроме того, цепочка может содержать символы, *не принадлежащие алфавиту.*

2 Реализация

2.1 Программная реализация

Для реализации были написаны два класса: **Analyser**, содержащий структуру построенного автомата и методы для анализа входной строки, и **Transition**, реализующий состояние автомата и возможные переходы из него. Кроме того, в файле **main.py** содержится реализация взаимодействия с пользователем. Подробное описание этих файлов представлено ниже.

2.1.1 Analyser.py

В конструкторе класса Analyser задаётся алфавит, над которым впоследствии будут строиться цепочки символов, а также создаются десять экземпляров класса Transition, соответствующие десяти состояниям, имеющимся в построенном конечном автомате (рис. 2). Эти экземпляры сохраняются в поле self.trans, к которому впоследствии будет происходить обращение для работы с автоматом. Начальное состояние принимает значение 1, а текст, поданный на вход, сохраняется в поле self.text в виде списка символов.
Вход: text - строка символов для анализа
Выход: конструктор не возвращает никакого значения

```
import random
from Transition import Transition as Tr

class Analyser:
    def __init__(self, text):
        self.text = list(text)
        self.current_state = 1
        self.alphabet = ['0', '1', '2', '3', '4', '5', '6', '7',
                         '8', '9', ':', ' ', 'a', 'p', 'm']
        self.range = self.alphabet[:-5]
        self.trans = [
            Tr(1, self.alphabet[:2], 2),
            Tr(1, list(self.alphabet[2]), 4),
            Tr(2, self.range, 3),
            Tr(3, list(self.alphabet[-5]), 6),
            Tr(4, self.alphabet[:4], 5),
            Tr(5, list(self.alphabet[-5]), 6),
            Tr(6, self.alphabet[:6], 7),
            Tr(7, self.range, 8),
            Tr(8, list(self.alphabet[-4]), 9),
            Tr(9, self.alphabet[-3:-1], 10),
            Tr(10, list(self.alphabet[-1]), 11)
        ]
```

Метод base_check() нацелен на то, чтобы проверить введённый текст на соответствие базовым правилам:

- текст не пуст;
- в тексте отсутствуют символы, не относящиеся к алфавиту;
- длина текста составляет ровно восемь символов.

В случае, если эти условия не выполняются, метод возвращает строку с сообщением об ошибке. В противном случае метод не возвращает никакого значения.
Вход: метод не принимает на вход никакого значения
Выход: строка - сообщение об ошибке

```
    def base_check(self):
        if len(self.text) == 0:
            return 'Empty string? Are you serious?..'
        for i in self.text:
            if i not in self.alphabet:
                return 'Unknown symbol: ' + i
        if len(self.text) != 8:
            return 'Fail...'
```

Метод analyse() извлекает очередной символ из входной строки и проверяет, возможен ли переход по этому символу из текущего состояния. В случае, если переход возможен, текущее состояние сменяется на то, в которое можно перейти, а затем метод вызывается рекурсивно для оставшейся части строки, либо завершает свою работу с кодом «Match!» (строка соответствует алфавиту), если строка пуста. В случае же, если такой переход из текущего состояния невозможен, метод завершает свою работу с кодом «Fail...» (строка не соответствует алфавиту).
Вход: метод не принимает на вход никакого значения
Выход: строка - сообщение о соответствии или несоответствии входной строки алфавиту

```
    def analyse(self):
        current_input = self.text.pop(0)
        marker = False
        for tr in self.get_possible_transition(self.current_state):
            if current_input in tr.get_transition():
                self.current_state = tr.get_final()
                marker = True
                break
        if not marker:
            return 'Fail...'
        if len(self.text) != 0:
            return self.analyse()
        else:
            return "Match!"
```

Метод get_possible_transition() принимает на вход номер состояния, в котором автомат находится на данный момент, и определяет, какие существуют возможные переходы из него. Возвращает список объектов Transition, в которых начальное состояние совпадает с поданным на вход.
Вход: state - номер текущего состояния
Выход: список объектов класса Transition

```
    def get_possible_transition(self, state):
        possible_transitions = []
        for tr in self.trans:
            if tr.get_base() == state:
```

```
                possible_transitions.append(tr)
        return possible_transitions
```

Метод get_random() реализует генерацию рандомной подходящей строки. Сперва начальное состояние устанавливается как 1, а поле self.text принимает значение пустой строки. После этого, до тех пор, пока поле self.current_state не примет значение 11 (финальное состояние), совершаются переходы по какому-то из возможных путей при помощи модуля random, текущее состояние изменяется, а в поле self.text добавлется случайный символ из числа тех, по которым мог быть совершен этот переход. Последнее реализуется посредством вызова метода get_random_transition() класса Transition. Метод возвращает значение self.text по итогу работы этого цикла.
Вход: метод не принимает на вход никакого значения
Выход: список символов - сгенерированная рандомная строка

```
    def get_random(self):
        self.current_state = 1
        self.text = ''
        while self.current_state != 11:
            possible_transitions = self.get_possible_transition(self.current_state)
            trans_number = random.randint(0, len(possible_transitions) - 1)
            self.text += possible_transitions[trans_number].get_random_transition()
            self.current_state = possible_transitions[trans_number].get_final()
        return self.text
```

2.1.2 Transition.py

Класс Transition содержит три поля:

- self.base_state - начальное состояние;
- self.transitions - список символов алфавита, по которым возможен переход из начального состояния в конечное;
- self.final_state - конечное состояние.

Для каждого из этих полей реализован метод-геттер.
Вход: base - номер исходного состояния, trans - список возможных символов для перехода, fin - номер финального состояния
Выход: конструктор не возвращает никакого значения

```
import random

class Transition:
    def __init__(self, base, trans, fin):
        self.base_state = base
        self.transitions = trans
        self.final_state = fin

    def get_base(self):
        return self.base_state
```

```
    def get_transition(self):
        return self.transitions

    def get_final(self):
        return self.final_state
```

Метод get_random_transition возвращает случайное значение из списка self.transitions при помощи модуля random.
Вход: метод не принимает на вход никакого значения
Выход: символ (строковый литерал), по которому возможен переход

```
    def get_random_transition(self):
        if len(self.transitions) == 0:
            return str(self.transitions[0])
        else:
            return str(self.transitions[random.randint(0, len(self.transitions) - 1)])
```

2.1.3 main.py

Файл main.py содержит код, организующий пользовательский ввод-вывод через консоль. Кроме того, в этом файле создается экземпляр класса Analyser, посредством которого производится работа с введённой строкой. Пользователю предлагается меню с тремя возможными операциями:

1. ввести текст для проверки;
2. сгенерировать рандомную строку;
3. выйти из программы.

```
from Analyser import Analyser

while True:
    print('Choose an option:\n
            1) Check your text\n
            2) Generate random matching string\n
            3) Quit')
    ans = input()
    if ans == '1':
        print('Enter text: ')
        text = input()
        analyser = Analyser(text)
        if analyser.base_check():
            print(analyser.base_check())
        else:
            print(analyser.analyse())
    elif ans == '2':
        analyser = Analyser('')
        print(analyser.get_random())
    else:
        break
```

3 Результат работы программы

Ниже представлены примеры работы программы. На рисунке 3 изображено пользовательское меню, с которого начинается работа программы. При выборе первого пункта пользователю предлагается ввести строку, после чего выводится сообщение «Match!», если она корректна (рис. 4) и «Fail...», если она некорректна (рис. 5). При вводе строки, содержащей символы не из алфавита, программа выдаёт сообщение об этом (рис. 6).
При выборе пункта 2 в меню на консоль выводится сгенерированная строка (рис. 7).

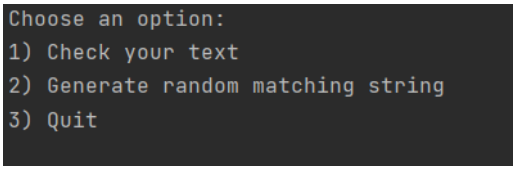

Рис. 3: Пользовательское меню

Рис. 4: Реакция программы на ввод корректного числа

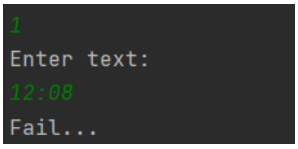

Рис. 5: Реакция программы на ввод некорректного числа

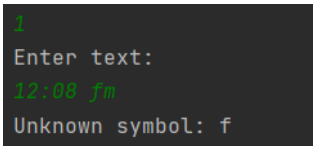

Рис. 6: Реакция программы на ввод символа не из алфавита

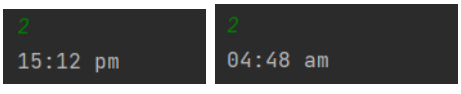

Рис. 7: Генерация рандомных строк

Заключение

В результате выполнения лабораторной работы была написана программа на языке Python, которая реализует конечный автомат, построенный по регулярному выражению для отображения времени в американском формате. Предусмотрен пользовательский ввод через консоль, анализ введённой строки, а также генерация случайной строки, подходящей под данный язык.

Достоинства программы

1. Обработка некорректного пользовательского ввода.
2. Уточнение информации о том, в каких лексемах входного файла допущены ошибки, при наличии таковых.

Недостатки программы

1. Алфавит задаётся списком, работа с которым производится по индексам - при добавлении в него символов потребуется реорганизация кода, описывающего возможные переходы.
2. Отсутствует возможность считывания строки из файла.

Список литературы

[1] Теория автоматов Ю.Г. Карпов - СПб.: Питер, 2003. - 208 с.

[2] Курс лекций по математической логике и теории автоматов. Востров А.В. https://tema.spbstu.ru/mathem/ (Дата последнего обращения: 04.06.2022)