МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Отчёт по дисциплине «Математическая логика»

Лабораторная работа №2 «Регулярные выражения» Вариант **№14**

Студент: _____ Салимли Айзек Мухтар Оглы

Преподаватель: _____ Востров Алексей Владимирович

«____»_____ 20___ г.

Санкт-Петербург, 2025

Содержание

Введение

В данном отчете, описана реализация программы, распазнающая ссылу на WEB страницу, по заданному варианту. Так же в отчете представлен конечный автомат, на основе которого спроектирована валидность ссылки, по регулярному выражению. Был спроектирован cabal проект. Программа была разделена на два .hs файла:

1. Lib.hs - Управляющая логика программы
2. Main.hs - Меню программы

Для реализации проекта, были выбраны:

- Cabal 3.0 - Сборщик проекта
- Haskell2010 - Спецификация языка
- Компилятор - GHC 9.12.1
- Haskell - Язык программирования
- VS Code - Среда разработки

Использованные библиотеки:

- base 4.19.2.0 - Стандартная библиотека Haskell
- random 1.2 - Для генерации ссылок
- text 1.2 - Для работы с текстом
- bytestring 0.10 - Для последующего использования в конечном автомате

1 Постановка задачи

По заданному варианту построить регулярное выражение, затем недетерминированный конечный автомат и детерминировать его (переходы можно задавать диапазонами). Реализовать программу, которая проверяет введенный текст через реализацию конечного автомата (варианты вывода: строка соответствует, не соответствует, символы не из алфавита). Также необходимо реализовать функцию случайной генерации верной строки по полученному конечному автомату.

Вариант №14:

- Проверка на соответствие ссылке яндекс диск

  Яндекс Диск, работает по нескольким ссылкам:

- disk.yandex.ru/d/... – часто встречающийся вид ссылки, ведущий на конкретный файл/папку.
- disk.yandex.ru/clients/disk/... – менее распространённый формат.
- yadi.sk/d/... – укороченный вариант публичной ссылки.

Реализованы все вышеперечисленные варианты.

2 Математическое описание

2.1 Конечный автомат

Программа работает на основе детерминированного конечного автомата, который определяется кортежем:

$$M = \{S, \Sigma, \delta, s_\theta, F\}$$

, где:

- Q - Конечное множество состояний
- Σ - Входной алфавит (ASCII символы)
- $\delta : S \times \Sigma \to Q$ - Функция переходов
- $s_\theta \in S$ - Начальное состояние
- $F \subseteq S$ - Множество финальных состояний

В коде множество состояний задается алгебраическим типом ADT с помощью data, и содержит:

$$Q = \{Start, H, Ht, Htt, Https, HttpsColon, HttpsColonSlash, HttpsColonSlashSlash,$$
$$D, Di, Dis, Disk, DiskDot, DiskDotY, DiskDotYa, DiskDotYan, DiskDotYand,$$
$$DiskDotYande, DiskDotYandex, YandexDot, YandexDotR, YandexDotRu,$$
$$YandexDotRuSlash, YandexDotRuSlashD, YandexDotRuSlashDSlash, Final\}$$

Входной алфовит можно определить как:

$$\Sigma = \{ASCII/(Control \cup Space)\}$$

Функция переходов определена в transition(в коде), можно описать: $\delta : S \times \Sigma \to S$

$$\delta(q, a) = \begin{cases} H & \text{если } q = Start,\ a = \text{'h'} \\ Ht & \text{если } q = H,\ a = \text{'t'} \\ Htt & \text{если } q = Ht,\ a = \text{'t'} \\ Https & \text{если } q = Htt,\ a = \text{'p'} \\ HttpsColon & \text{если } q = Https,\ a = \text{'s'} \\ HttpsColonSlash & \text{если } q = HttpsColon,\ a = \text{':'} \\ HttpsColonSlashSlash & \text{если } q = HttpsColonSlash,\ a = \text{'/'} \\ D & \text{если } q = HttpsColonSlashSlash,\ a = \text{'/'} \\ \dots & \text{(по аналогии, вплоть до Final)} \\ Final & \text{если } q = YandexDotRuSlashDSlash,\ a \in \Sigma_v \\ q & \text{если } a \in \text{Space} \\ Start & \text{иначе} \end{cases}$$

где $\Sigma_v = \{c \in \Sigma \mid c \notin \text{Control} \cup \text{Space}\}$ — допустимые символы пути.

- $s_\theta = Start$ — начальное состояние
- $F = \{Final\}$ — множество финальных состояний

Начальное состояние: $s_0 = Start$

Финальное состояние: $F = Final$

2.2 Язык распознаваемый автоматом

Автомат распознаёт язык $L \subseteq \Sigma^*$, содержащий строки следующего вида:

$$L = \{w = p \cdot s \mid p \in P,\ s \in \Sigma_v^+\}$$

, где P = https://disk.yandex.ru/d/..., https://disk.yandex.ru/clients..., https://yadisk.ru/d/...

То есть, строка должна начинаться с допустимого префикса и содержать непустую последовательность валидных символов после него.

2.3 Граф конечного автомата

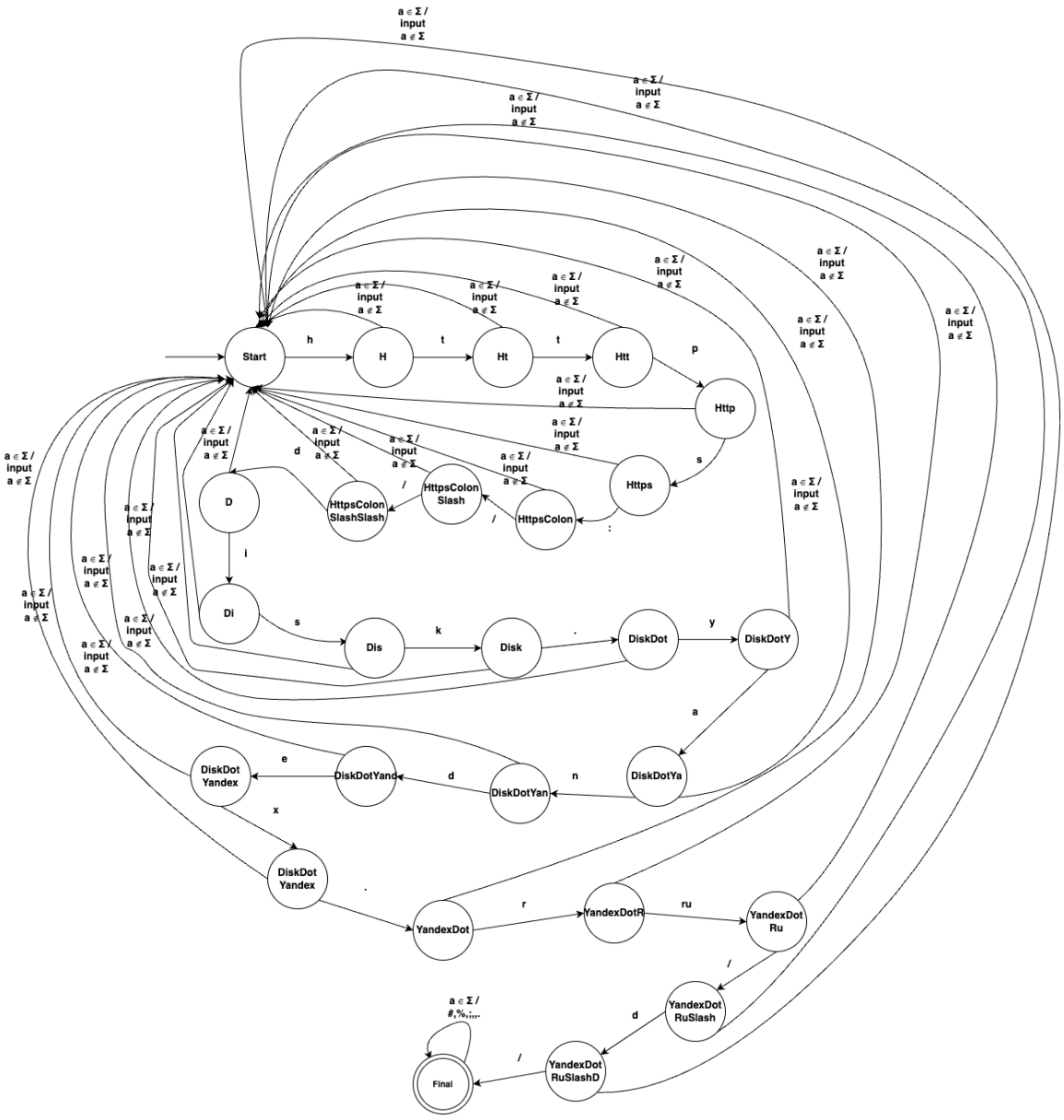

Рис. 1: Граф конечного автомата

3 Программная реализация

В качестве языка программирования был выбран Haskell
В качестве стандарта языка, был выбран Haskell2010
Для реализации программы, был собран cabal - проект, который включает в себя два .hs файла:

- Main.hs - Отвечающий за меню программы
- Lib.hs - Отвечающий за управляющую логику

В листинге 1, представлен код Main.hs. В листинге 2, представлен код Lib.hs:

3.1 Main.hs

Листинг 1: Main.hs

```
module Main where
import Lib

        main :: IO ()
        main = do
                putStrLn "Select an option:"
                putStrLn ""
                putStrLn "1. Check the link"
                putStrLn ""
                putStrLn "2. Generate a valid link"
                putStrLn ""
                putStrLn "3. Show link pattern"
                putStrLn ""
                putStrLn "4. Exit"
                putStrLn ""
                putStrLn "Enter the number and press Enter:"
                choice <- getLine

                case choice of
                        "1" -> do
                                putStrLn ""
                                putStrLn "Enter the link to check:"
                                putStrLn ""
                                input <- getLine
                                checkYandexDiskLink input
                                main
                        "2" -> do
                                putStrLn ""
                                link <- generateValidLink
                                putStrLn $ "Generated link: " ++ link
                                putStrLn ""
                                main
                        "3" -> do
                                putStrLn ""
                                putStrLn $ "Pattern /d/: " ++ "https://disk.
                                    yandex.ru/d/..."
                                putStrLn $ "Pattern /clients/: " ++ "https
                                    ://yadi.sk/d/..."
                                putStrLn $ "Pattern /clients/disk/: " ++ "
                                    https://disk.yandex.ru/clients/disk/..."
                                putStrLn ""
                                main
                        "4" -> putStrLn "Goodbye!"
                        _ -> do
                                putStrLn ""
```

```
                                    putStrLn ""
                                    putStrLn "Invalid choice, please try again"
                                    putStrLn ""
                                    putStrLn ""
                                    main
```

- Монада IO, выводит меню
- Обертка do, вызывает кейс для выбора подпрограмм программы, где:
  1. Проверка строки на соответствие ссылки Яндекс Диска

     Вызывает функцию checkYandexDiskLink

  2. Генератор валидной ссылки Яндекс Диска

     Вызывает функцию generateValidLink и перечисляет его в лист

  3. Демонстрация шаблонов

     Выводит шаблоны в консоль среды разработки

  4. Завершение программы

     Завершает программу и выводит сообщение

При ином входе в main функцию, выводится сообщение о неверном выборе, затем происходит возврат в меню.

3.2 Lib.hs

Управляющая логика, основана на работе конечного автомата, на вход которого поступают введенные символы, далее функция переходов, переходи в следующее состояние, и так пока состояние не будет Final, либо пока не будет введен некорректный символ.

Листинг 2: Lib.hs

```
        module Lib
    ( checkYandexDiskLink
    , generateValidLink
    ) where

import Data.Char (isAscii, isSpace, isControl)
import System.Random (randomRIO)
import Control.Monad (replicateM)
import Data.List (isInfixOf)

validChars :: String
validChars = ['a'..'z'] ++ ['0'..'9'] ++ "-_"

preprocess :: String -> String
preprocess = filter (not . isSpace) . map toLower
  where
    toLower c =
      if c >= 'A' && c <= 'Z'
      then toEnum (fromEnum c + 32)
      else c

data State = Start
           | H | Ht | Htt | Https
           | HttpsColon | HttpsColonSlash | HttpsColonSlashSlash
           | D | Di | Dis | Disk
           | DiskDot | DiskDotY | DiskDotYa | DiskDotYan | DiskDotYand
```

```
            | DiskDotYande | DiskDotYandex
            | YandexDot | YandexDotR | YandexDotRu
            | YandexDotRuSlash
            | YandexDotRuSlashD
            | YandexDotRuSlashDSlash
            | YandexDotRuSlashC
            | YandexDotRuSlashCl
            | YandexDotRuSlashCli
            | YandexDotRuSlashClie
            | YandexDotRuSlashClien
            | YandexDotRuSlashClient
            | YandexDotRuSlashClients
            | YandexDotRuSlashClientsSlash
            | YandexDotRuSlashClientsSlashD
            | YandexDotRuSlashClientsSlashDi
            | YandexDotRuSlashClientsSlashDis
            | YandexDotRuSlashClientsSlashDisk
            | YandexDotRuSlashClientsSlashDiskSlash
            | Y | Ya | Yad | Yadi
            | YadiDot | YadiDotS | YadiDotSk
            | YadiDotSkSlash
            | YadiDotSkSlashD
            | YadiDotSkSlashDSlash
            | Final
            deriving (Eq, Ord, Show, Enum, Bounded)

isValidChar :: Char -> Bool
isValidChar c = not (isSpace c || isControl c)

transition :: State -> Char -> State
transition Start 'h'                = H
transition H 't'                    = Ht
transition Ht 't'                   = Htt
transition Htt 'p'                  = Https
transition Https 's'                = HttpsColon
transition HttpsColon ':'           = HttpsColonSlash
transition HttpsColonSlash '/'      = HttpsColonSlashSlash
transition HttpsColonSlashSlash 'd' = D
transition HttpsColonSlashSlash 'y' = Y
transition Start 'd' = D
transition Start 'y' = Y
transition D 'i' = Di
transition Di 's' = Dis
transition Dis 'k' = Disk
transition Disk '.' = DiskDot
transition DiskDot 'y'      = DiskDotY
transition DiskDotY 'a'     = DiskDotYa
transition DiskDotYa 'n'    = DiskDotYan
transition DiskDotYan 'd'   = DiskDotYand
transition DiskDotYand 'e'  = DiskDotYande
transition DiskDotYande 'x' = DiskDotYandex
transition DiskDotYandex '.' = YandexDot
transition YandexDot 'r'       = YandexDotR
transition YandexDotR 'u'      = YandexDotRu
transition YandexDotRu '/' = YandexDotRuSlash
transition YandexDotRuSlash 'd' = YandexDotRuSlashD
transition YandexDotRuSlash 'c' = YandexDotRuSlashC
transition YandexDotRuSlashD ' ' = YandexDotRuSlashD
transition YandexDotRuSlashD '/' = YandexDotRuSlashDSlash
transition YandexDotRuSlashDSlash _ = Final
transition YandexDotRuSlashC 'l' = YandexDotRuSlashCl
```

```haskell
transition YandexDotRuSlashCl 'i' = YandexDotRuSlashCli
transition YandexDotRuSlashCli 'e' = YandexDotRuSlashClie
transition YandexDotRuSlashClie 'n' = YandexDotRuSlashClien
transition YandexDotRuSlashClien 't' = YandexDotRuSlashClient
transition YandexDotRuSlashClient 's' = YandexDotRuSlashClients
transition YandexDotRuSlashClients '/' = YandexDotRuSlashClientsSlash
transition YandexDotRuSlashClientsSlash 'd' = YandexDotRuSlashClientsSlashD
transition YandexDotRuSlashClientsSlashD 'i' =
   YandexDotRuSlashClientsSlashDi
transition YandexDotRuSlashClientsSlashDi 's' =
   YandexDotRuSlashClientsSlashDis
transition YandexDotRuSlashClientsSlashDis 'k' =
   YandexDotRuSlashClientsSlashDisk
transition YandexDotRuSlashClientsSlashDisk '/' =
   YandexDotRuSlashClientsSlashDiskSlash
transition YandexDotRuSlashClientsSlashDiskSlash _ = Final
transition Y 'a'      = Ya
transition Ya 'd'     = Yad
transition Yad 'i'    = Yadi
transition Yadi '.'   = YadiDot
transition YadiDot 's' = YadiDotS
transition YadiDotS 'k' = YadiDotSk
transition YadiDotSk '/' = YadiDotSkSlash
transition YadiDotSkSlash 'd' = YadiDotSkSlashD
transition YadiDotSkSlashD '/' = YadiDotSkSlashDSlash
transition YadiDotSkSlashDSlash _ = Final

transition Final c
    | isValidChar c = Final
    | otherwise     = Start
transition s ' '
    | s /= Final  = s
    | otherwise   = Final
transition _ _ = Start

checkYandexDiskLink :: String -> IO ()
checkYandexDiskLink input = do
    if not (all isAscii input)
        then putStrLn "Oops! Only ASCII symbols!"
        else do
            let processed = preprocess input
            if isMainPage processed
                then putStrLn "Yandex Disk main page"
                else do
                    let result = processString processed Start
                    if result == Final && hasValidPath processed
                        then putStrLn "String - is a Yandex Disk link"
                        else putStrLn "Oops! Not a Yandex Disk link"
  where
    isMainPage :: String -> Bool
    isMainPage s =
        let normalized = preprocess s
        in  normalized == "https://disk.yandex.ru"
         || normalized == "disk.yandex.ru"
         || normalized == "https://disk.yandex.ru/"
         || normalized == "disk.yandex.ru/"
         || normalized == "disk.yandex.ru/clients/"
         || normalized == "disk.yandex.ru/clients/disk/"
         || normalized == "yadi.sk"
         || normalized == "yadi.sk/"

```

```haskell
    processString :: String -> State -> State
    processString [] st     = st
    processString (c:cs) st = processString cs (transition st c)

    hasValidPath :: String -> Bool
    hasValidPath s
      | "disk.yandex.ru/d/" `isInfixOf` s = True
      | "disk.yandex.ru/clients/disk/" `isInfixOf` s = True
      | "yadi.sk/d/" `isInfixOf` s = True
      | otherwise = False

generateValidLink :: IO String
generateValidLink = do
    idLength <- randomRIO (12, 20)
    idPart   <- replicateM idLength (randomChar validChars)
    patternIndex <- randomRIO (1 :: Int, 3)
    return $ case patternIndex of
        1 -> "https://disk.yandex.ru/d/" ++ idPart
        2 -> "https://disk.yandex.ru/clients/disk/" ++ idPart
        3 -> "https://yadi.sk/d/" ++ idPart
        _ -> "https://disk.yandex.ru/d/" ++ idPart
  where
    randomChar :: String -> IO Char
    randomChar chars = do
        idx <- randomRIO (0, length chars - 1)
        return (chars !! idx)
```

Функция preprocess

- Тип: `String -> String`
- На входе: строка исходного текста (например, введённая ссылка)
- На выходе: строка без пробелов (и приведённая к нижнему регистру)
- Нужна для: очистки входной строки перед проверкой или генерацией ссылки

Функция checkYandexDiskLink

- Тип: `String -> IO ()`
- На входе: строка, содержащая ссылку
- На выходе: IO-экшн, который выводит сообщение о результате проверки
- Нужна для: проверки, является ли введённая строка валидной ссылкой Яндекс Диска

Функция generateValidLink

- Тип: `IO String`
- На входе: отсутствует (генерируется случайным образом)
- На выходе: сгенерированная строка, представляющая валидную ссылку Яндекс Диска
- Нужна для: автоматической генерации валидных ссылок

Функция transition

- Тип: `State -> Char -> State`
- На входе: текущее состояние автомата и входной символ (`Char`)
- На выходе: новое состояние автомата

- Нужна для: реализации функции переходов конечного автомата, определяющей, в какое состояние перейти для данного символа

Функция processString

- Тип: `String -> State -> State`
- На входе: строка для обработки и начальное состояние автомата
- На выходе: конечное состояние автомата после обработки всей строки
- Нужна для: последовательного применения функции `transition` ко всем символам входной строки

Заключение

В результате выполнения лабораторной работы, был реализован cabal-проект, содержащий код программы, работающий на логике конечного автомата, который проверяет регулярное выражение - ссылку на Яндекс Диск. Были реализованы три варианта обращения к сайту:

- https://disk.yandex.ru/d/...
- https://disk.yandex.clients/d/...
- https://yadisk.ru/d/...

Плюсы:

- Конечный автомат позваляет четко проверять правильность ссылки
- Функция генерации всегда выдает валидные ссылки, так же легко изменяема
- Была реализована функция преведения ссылки в нормальную форму (без пробелов)

Минусы:

- Большое количество состояний и переходов сказывается на скорости работы программы
- Любое изменение формата ссылки требует правок во многих правилах переходов, что увеличивает риск ошибок
- Отсутствие явного логгирования

Масштабируемость:

- Добавить явный вывод, что не так во входной ссылке
- Реализация проверок иных сайтов Яндекса
- Разбиение обработок на модлуи для разных ссылок

Список литературы

1. Востров, А. В. Математическая логика [Электронный ресурс]. Режим доступа: https://tema.spbstu.ru/compiler/ (последний визит: 01.04.2025).
2. Сети, Р.; Ахо, А. Компиляторы: принципы, технологии и инструменты / Р. Сети, А. Ахо. – М.: Издательство «Наука», 2006. – С. 104.