Формальные языки, занятие 1

Термины

DFA (конечные автоматы), распознавание строк, языки, состояния, функция перехода, начальное состояние, терминальные состояния – смотрите лекцию.

“Класс замкнут относительно операции” – означает, что если имеется множество X и операция f, которая принимает какое-то количество X, то результат f лежит в X. Например, натуральные числа замкнуты относительно сложения: сложение двух натуральных чисел всегда даёт натуральное число; натуральные числа не замкнуты относительно вычитания, так как $3 - 6 = -3$ не является натуральным числом.

Способы записи конечных автоматов

Словами

Всё должно быть ясно.

В виде схем

В виде программ

```haskell
{-# LANGUAGE RecordWildCards #-}
import Data.List (foldl')

-- |Описание конечных автоматов с состоянием `s` и алфавитом `g`.
data DFA s g = DFA
```

```
  { -- |Функция перехода
    delta :: g -> s -> s
  , -- |Предикат, пределяющий терминальные состояния
    isTerminal :: s -> Bool
  , -- |Начальное состояние
    initState :: s
  }

-- |Функция, которая проверяет, распознаётся ли данная строка данным DFA.
runDFA :: DFA s g -> [g] -> Bool
runDFA DFA {..} = isTerminal . foldl' delta initState
```

Разбор задач

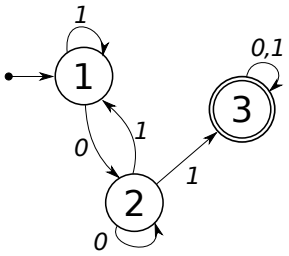
Язык слов, содержащих подстроку “01”

Вводим три состояния:

1. Состояние, в котором последний символ не 0, а также ещё не была встречена строка “01”.
2. Состояние, в котором последний символ 0, а также строка “01” ещё не была встречена.
3. Состояние, где строка “01” уже была встречена.

Начальное состояние №1; терминальных состояний только одно, и это №3.

Переходы: из состояния №1 в состояние №2 можно перейти по символу 0; из состояний №1 и №2 в состояние №1 – по символу 1; из состояния №2 в №3 – по символу 1. Корректность таких переходов должна быть понятна по их определению.




```
data P1 = P1Init  -- 1
        | P1Last0 -- 2
        | P1Met01 -- 3
        deriving (Eq, Show)

-- True = 1, False = 0
-- |Конечный автомат, распознающий только строки, включающие "01".
```

```haskell
p1 :: DFA P1 Bool
p1 = DFA {..}
  where delta P1Init  True  = P1Init
        delta P1Init  False = P1Last0
        delta P1Met01 _     = P1Met01
        delta P1Last0 True  = P1Met01
        delta P1Last0 False = P1Last0
        isTerminal = (==) P1Met01
        initState = P1Init
```

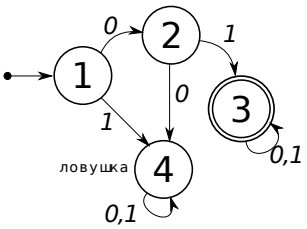
Язык слов, начинающихся с подстроки “01”

Четыре состояния:

1. Начальное состояние;
2. Состояние, где была принята строка “0”;
3. Состояние, где была принята строка “01” и, может, ещё какой-то произвольный набор символов.
4. Состояние, в котором уже ясно, что строка не будет распознана.

Начальное состояние по определению №1; терминальное – №3.

Переходы: из №1 по 0 переход в №2, из №2 по 1 переход в №3; из №3 по любому символу можно попасть в себя же; все остальные переходы напрявляются в состояние №4, из которого не выбраться.



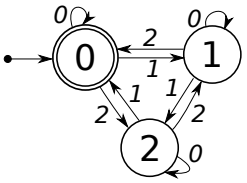

```haskell
data P2A = P2AInit
         | P2AMet0
         | P2AMet01
         | P2AReject
         deriving (Eq, Show)

p2a :: DFA P2A Bool
p2a = DFA {..}
  where delta P2AInit  False = P2AMet0
        delta P2AMet0  True  = P2AMet01
        delta P2AMet01 _     = P2AMet01
        delta _        _     = P2AReject
        isTerminal = (==) P2AMet01
        initState = P2AInit
```

Язык слов в алфавите {0, 1, 2} таких, что сумма цифр делится на 3

Три состояния, каждое представляет остаток по модулю 3 суммы цифр в считанном числе.

Начальное состояние – это состояние, соответствующее 0. Оно же является единственным терминальным.

Переходы: 0 (+1) -> 1, 0 (+2) -> 2, 1 (+1) -> 2, 1 (+2) -> 0, 2 (+1) -> 0, 2 (+2) -> 1, где “+x” означает разом и считывание символа “x”, и происходящую при этом арифметическую операцию над суммой цифр по модулю 3. Из всех состояний можно попасть в себя же по нулю.




```haskell
data P3 = P3_0
        | P3_1
        | P3_2
        deriving (Eq, Show)

p3 :: DFA P3 Int -- алфавит неточный
p3 = DFA {..}
  where delta P3_0 0 = P3_0
        delta P3_0 1 = P3_1
        delta P3_0 2 = P3_2
        delta P3_1 0 = P3_1
        delta P3_1 1 = P3_2
        delta P3_1 2 = P3_0
        delta P3_2 0 = P3_2
        delta P3_2 1 = P3_0
        delta P3_2 2 = P3_1
        delta _    _ = error "Лень делать трёхэлементный алфавит"
        isTerminal = (==) P3_0
        initState = P3_0
```

Язык $a^n b^n$ не распознаётся конечным автоматом

Заметим, что переход из некоторого состояния по некоторой строке всегда происходит одинаково; *всё* состояние конечного автомата определяется его текущим состоянием.

Покажем, что нам не хватит никакого конечного количества состояний, чтобы распознать алфавит $a^n b^n$.

Пусть мы считали n букв “a” и они закончились, дальше пойдут только буквы “b”. Значит, мы должны сейчас находиться в состоянии, при проходе по которому можно распознать только b^n, но не b^m для $n \neq m$. Назовём такое состояние s_n.

Докажем теперь от противного: пусть кто-то предоставил конечный автомат, который распознаёт ровно строки $a^n b^n$. Пусть в нём k состояний. Если теперь мы подадим ему на вход a^r, где $r > k$, то окажемся в состоянии s_r. Но заметим теперь, что множество $\{s_i \mid i \in [0; r]\} \subseteq S$ по мощности меньше, чем k, как подмножество S, а раз так, то выполняется $s_i = s_j$ для каких-то $i \neq j$.

Такого быть не может: при переходах из s_i можно распознать только b^i, а при переходах из s_j – только b^j.