Дз №1 по Формальным языкам

Плотников Даниил Викторович

12 сентября 2021 г.

1

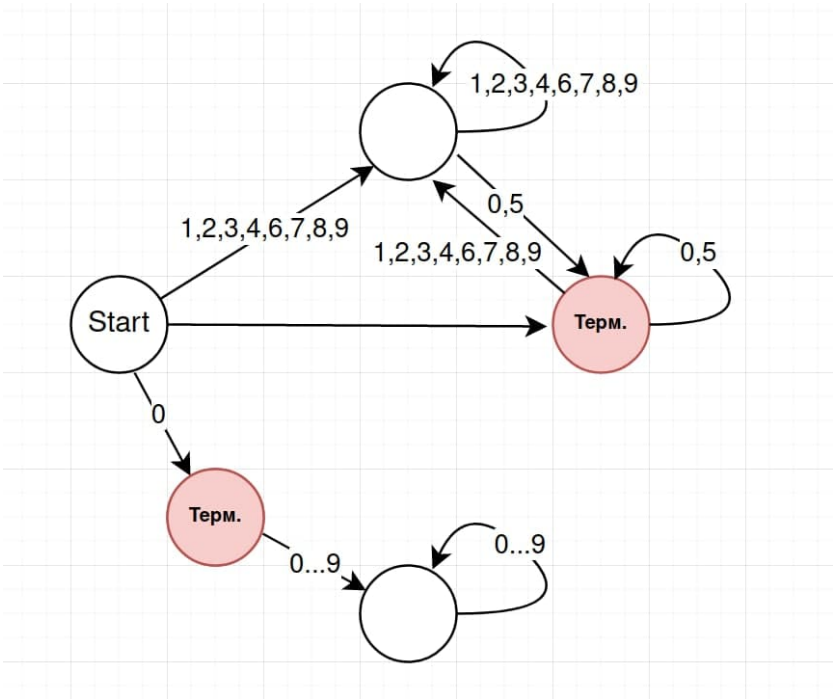

Рис. 1: Полный конечный автомат, проверяющий, что данное на вход чисто будет делиться на 5 и не иметь незначащие нули

2

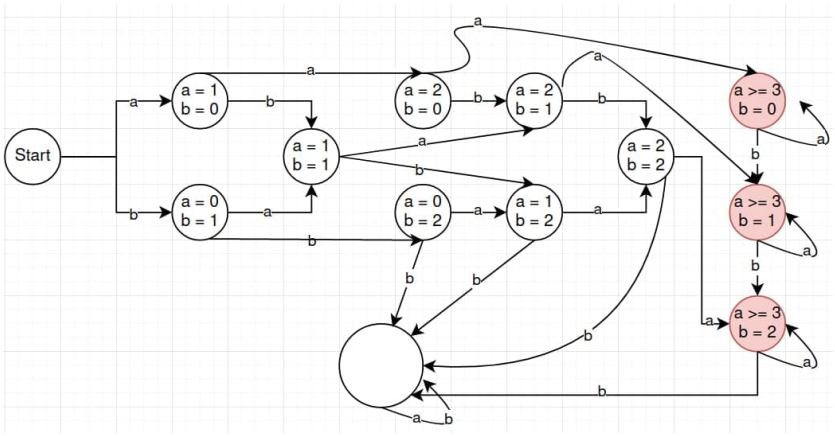

Рис. 2: Полный конечный автомат, проверяющий, что данное на вход последовательность содержит не более 2 литералов{b} и более 2 литералов {a}

3

В качестве языка программирования я буду рассматривать язык GO. Приведу пару особенностей лексичесткого синтаксиса, о которых я не знал.

1. Начнём с типа данных, о котором существование которого я даже не догадывался. Есть отдельный тип данных, при помощи которого можно задавать комплексные числа. Спецификация: тык

```
complex64
complex128
b := complex(6, 4)
```

2. Константы можно объединять в набор связных, а также использование *iota* позволяет задать значение константы, которое будет равно номеру в табом наборе. Спецификация: тык

3. Так как *Go* поддерживает *Unicode*, то можно использовать русские буквы в названии (ещё +1 способ выстрелить себе в ногу: собес -> собес, теперь сложно догататься литералы с, о, е английские или русские). Спецификация: тык

4. Т.к. функции могут возвращать несколько элементов, то может случится так, что нам некоторые переменные не нужны, в Go можно их проигнорировать. Спецификация: тык

```
x, _ = f()   // evaluate f() but ignore second result value
```

5. В *switch* конструкции можно передавать управлению следующему пункту case. Это можно делать при помощи $fallthrough$. Спецификация: тык

6. Именнованные результирующие параметры в возвращаемых значениях функции. Мы можем сразу задать имя той переменной, которую ходим вывести и просто вызвать *return* без аргументов и функция возвратит n. Спецификация: тык

7. defer функции - фукции, которые начинают выполняться, когда область видимости, в которой они находятся, будет закончена(то есть все функции в данной области будут закончены). Выполняются в обратном порядке вызова их. Спецификация: тык

4

Язык:

1. Алфавит: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, запятая, точка с запятой, тире$\}$

2. Для удобства пометим вершины: начальная вершина $= 0$, обычная вершина $= 1$, терминальная $= 2$, сток $= 3$.

3. Каждый конечный автомат представим в виде вершин и рёбер. Число = последовательность цифр, между которыми нет других символов. А ребро будем определять так $1, 3 : 1, 2, 3$, из вершины 1 можно попасть в вершину 3, если на входе будет одно из трёх чисел: 1, 2, 3. Все рёбра и вершины (когда определяем их состояние) разделены ; . Сначала идёт чисто = количество вершин, потом

сами вершины и номера, которые определяют состояние этой вершины(начальная, обычная, терминальная сток). Пусть вершина 1 будет начальная всегда, а последняя стоком. Дальше идут рёбра.

4. Задача 1: Определить, является число нулём без лидирующих нулей?

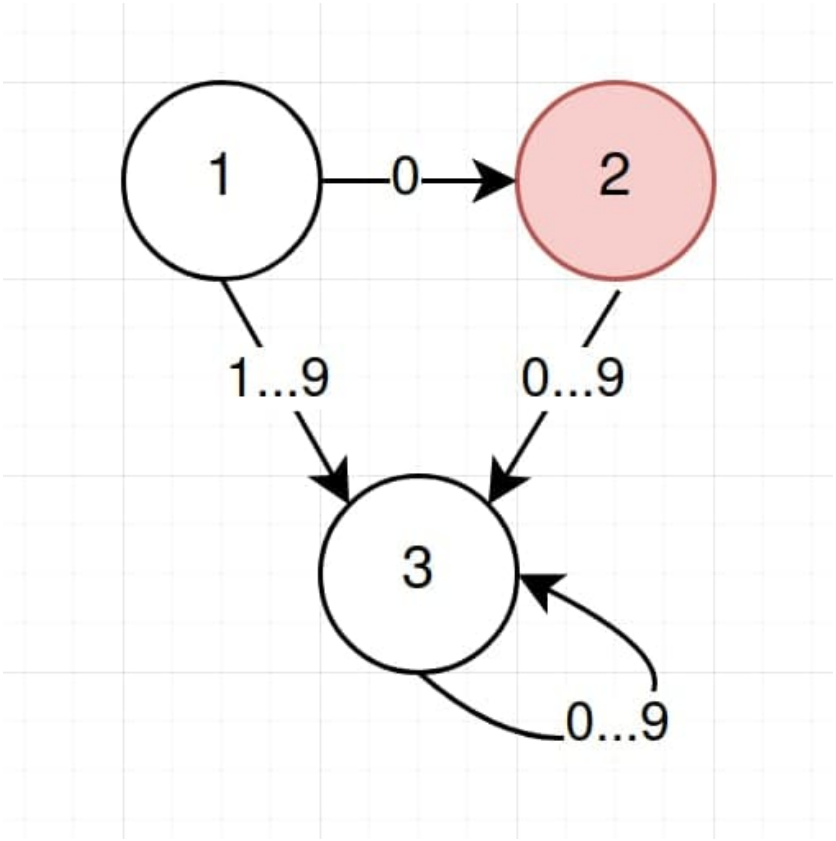

Рис. 3:

Запись на описанном выше языке:
3;1,0;2,2;3,3;1,2:0;1,3:1,2,3,4,5,6,7,8,9;2,3:0,1,2,3,4,5,6,7,8,9;3,3:0,1,2,3,4,5,6,7,8,9

5. Задача 2: На вход даётся неотрицательное число(лидирующие нули могут быть). Определить: меньше оно числа 2?

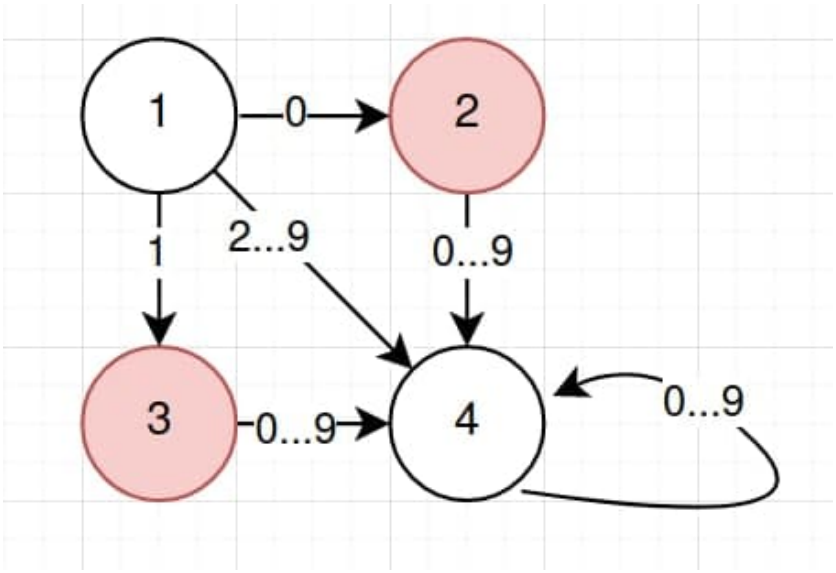

Рис. 4:

Запись на описанном выше языке:
4;1,0;2,2;3,2;4,3;1,2:0;1,3:1;1,4:2..9;2,4:0..9;3,4:0..9;4,4:0..9

6. Задача 3: На вход даётся неотрицательное число(лидирующие нули могут быть). Определить: чётно ли число?

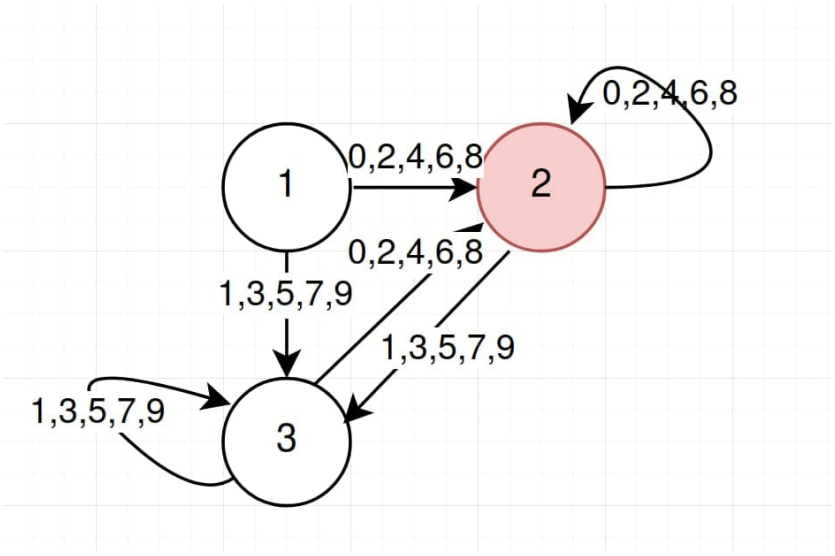

Рис. 5:

Запись на описанном выше языке:
3;1,0;2,2;3,1;1,2:0,2,4,6,8;1,3:1,3,5,7,9;2,2:0,2,4,6,8;2,3:1,3,5,7,9;3,2:0,2,4,6,8;3,3:1,3,5,7,9

5

```
1  import scala.io._
2  import scala.util.parsing.json._
3  import java.net._
4
5  object Twitsearch {
6      // comment
7      Int one_element = 1000;
8    def main(args: Array[String]) {
9      if(args.isEmpty) {
10       println("Usage: twitsearch <query>")
11       return
12     }
13     val query = URLEncoder.encode(args(0), "UTF-8")
14     val url = "http://search.twitter.com/search.json?q=" + query
15     val data = Source.fromURL(url).mkString
16
17     JSON.parseFull(data) match {
18       case None =>
19         println("Failed to parse JSON")
20       case Some(j) =>
21         val json = j.asInstanceOf[Map[String,List[Map[String,_]]]]
22         json("results").map(x => x("text")).foreach(println)
23     }
24   }
25 }
```

Рис. 6: Результат выполнения подсветки синтаксиса