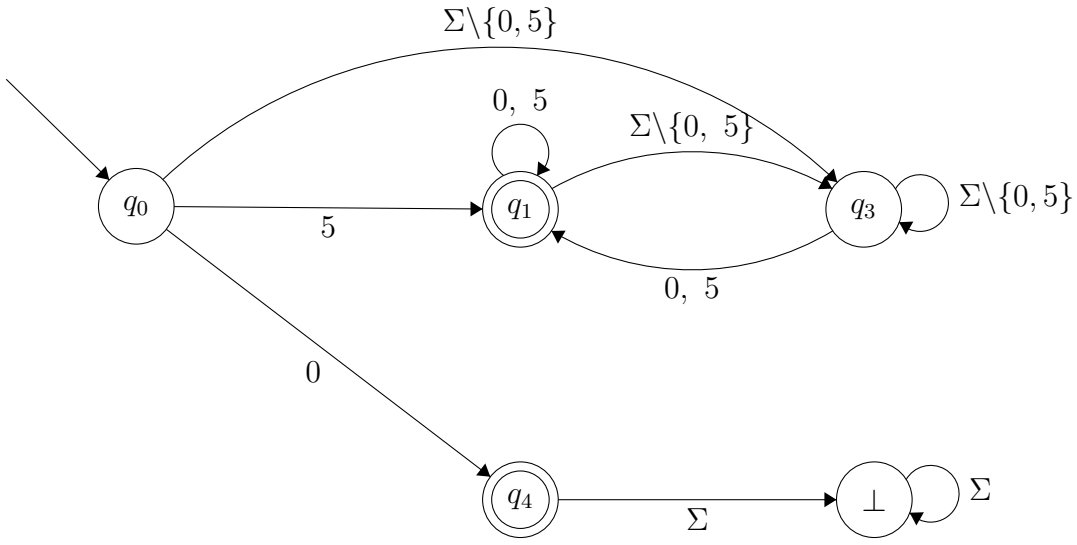
Домашняя работа

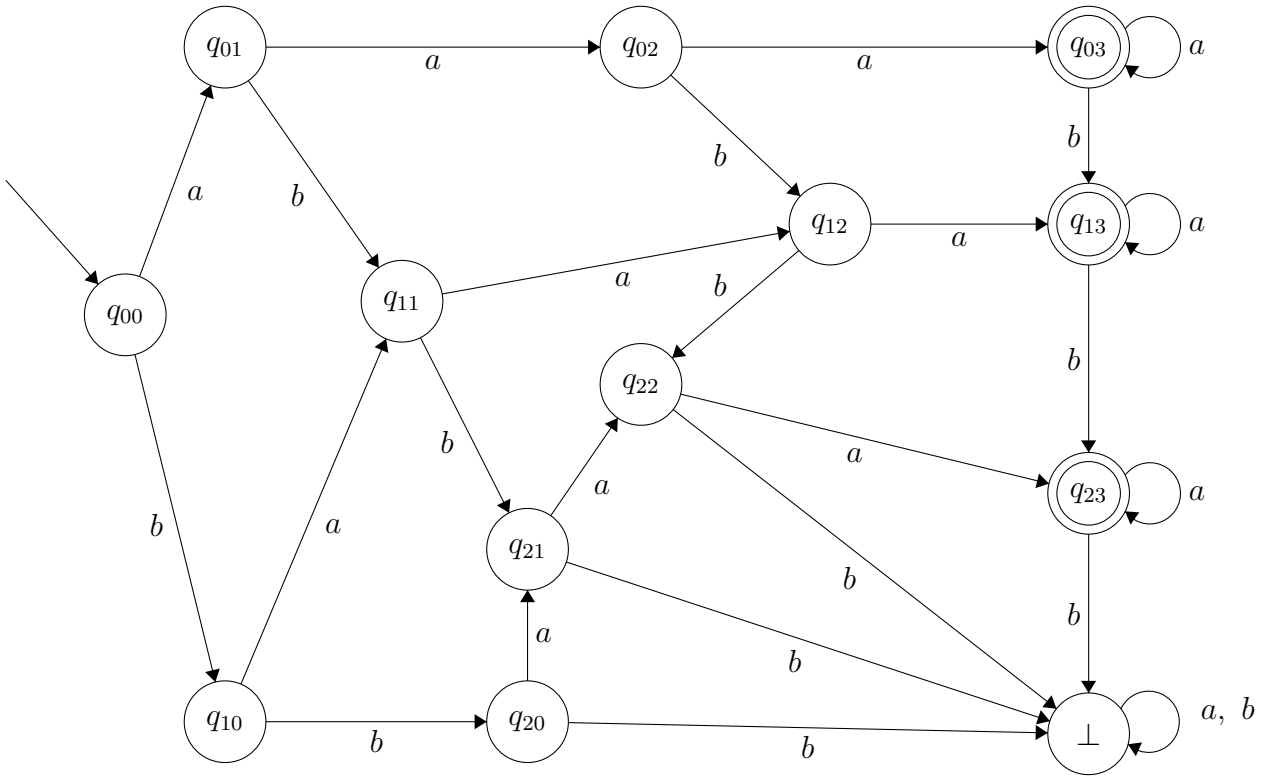
Суворов Вячеслав

18 сентября 2021 г.

1. Начальная вершина q_0



2. q_{cd} d показывает, сколько букв a. c показывает, сколько букв b, причем если d == 3, то это значит что букв a больше или равно трем. Начальная вершина q_{00}



3. Для описания лексического синтаксиса, я выбрал язык Kotlin.
Можно кликнуть по названия пункта и перейти на официальную документацию.

   (a) The !! operator
   Я узнал, что существует особый оператор, который помогает нам обрабатывать и избегать проблемы с Null Pointer Exception. Это оператор !!. Если мы хотим получить ненулевой тип, или получить extension то мы можем использовать его. В примере по ссылке можно увидеть: val l = b!!.length. Это бывает очень полезно, когда мы хотим проверить ввод пользователя

   (b) "Safe" (nullable) cast operator
   Стоит сказать, что после некоторых типов к котлине мы можем написать символ вопроса ('?') этот символ помогает нам понять, лежит ли тм тип, или null. Если этот опертор не присовить, будет ошибка компиляции. Но есть опертор безопасного приведения типов.
   val x: String? = y as? String
   В таком случае резкльтат nullable

   (c) noinline
   В языке Kotlin есть возможность использование всстроенные лямбды в инлайн функции, то у нас есть такая возможность, написав перед ними протсо слово noinline без использования никаких других функций.

   (d) Data classes
   Удобно, что есть синтаксический сахар над обычными классами, вместо того, чтобы писать полностью свой класс можно использовать их и автоматичски будут написаны некоторые функции: equals+hashCode+toString()

(e) Unary operations
Вместо унарных операторов можно указывать конкретные методы. если на плюсах нужно указывать ключевое слово, то теперь можн опереопредять их с помощью имени метода, вот пример: operator fun Point.unaryMinus() = Point(-x, -y)

4. Для описания конечного автомата будем использовать русский язык (он является естественным языком). Сначала мы зададим все вершины, причем самой первой записывается начальная вершина, потом все вершины остальные. Просто записываем имя вершины. Если вершина терминальная, то справа от нее пишем терминальная. Все вершины кладем внутри блока ВЕРШИНЫ НАЧАЛИСЬ *вершины* ВЕРШИНЫ КОНЧИЛИСЬ. Теперь зададим все ребра в блоке РЕБРА НАЧАЛИСЬ *все ребра* РЕБРА КОНЧИЛИСЬ.
Записываем ребро так:
'из какой вершины', 'в какую вершину', {'перечисляем по каким символам'};
Записываем вершину так: 'имя', 'ялвляется ли вершина терминальной ДА/НЕТ';
Примеры:

Является ли строка строкой ab (алфавит a, b, c)
ВЕРШИНЫ НАЧАЛИСЬ
q_0, НЕТ;
q_1, НЕТ;
q_2, ДА;
$\perp$, НЕТ;
ВЕРШИНЫ КОНЧИЛИСЬ
РЕБРА НАЧАЛИСЬ
q_0, q_1, {a};
q_0, $\perp$, {b};
q_1, q_2, {b};
q_1, $\perp$, {a};
q_2, $\perp$, {a, b};
$\perp$, $\perp$, {a, b};
РЕБРА КОНЧИЛИСЬ

Является ли число корректным (без ведущих нулей) (алфавит 0..9)
ВЕРШИНЫ НАЧАЛИСЬ
q_0, НЕТ;
q_1, ДА;
q_2, ДА;
$\perp$, НЕТ;
ВЕРШИНЫ КОНЧИЛИСЬ
РЕБРА НАЧАЛИСЬ
q_0, q_1, {1..9};
q_0, q_2, {0};
q_1, q_1, {0..9};
q_2, $\perp$, {0..9};
$\perp$, $\perp$, {0..9};
РЕБРА КОНЧИЛИСЬ

Заканчивается ли число на 1 (алфавит 0, 1)
ВЕРШИНЫ НАЧАЛИСЬ
q_2, НЕТ;
q_0, НЕТ;
q_1, ДА;
ВЕРШИНЫ КОНЧИЛИСЬ
РЕБРА НАЧАЛИСЬ
q_2, q_1, {1};
q_2, q_0, {0};
q_1, q_0, {0};
q_1, q_1, {1};
q_0, q_1, {1};
q_0, q_0, {0};
РЕБРА КОНЧИЛИСЬ

5. я делал подсветку кода в Sublime. Для того, чтобы применить ее, зайдите в /.config/sublime-text/Packeges/User и в эту папку закиньте файл $My\_syntax.sublime-syntax$. Потом создайте любой файл и приминте мою подсветку View->Syntax->$My_syntax.sublime-syntax$v
Я посвечивал строковые литерал, так как без подсвтеки и они могут сливаться с остальный кодом. Цифры я подсветил чтобы было удобно разделить из и цифры из имен переменных. Аналогичную вещь я сделал и с именами переменных. Логические операторы я подсветил, так как люблю при записи длинных логичских выражений видеть разницу. Были подсвечены и ключевые слова, чтобы сразу видеть циклы, условия и так далее. Типы я тоже подветил, чтобы сразу обращать внимание на происходящее. Подсветил TODO как и в большинстве IDE, чтобы при промотке был контраст и юзер видел что осталось сделать.

```
%YAML 1.2
---
# See http://www.sublimetext.com/docs/syntax.html
file_extensions:
  - ec
scope: source.example-c
contexts:
  main:
    #
    - match: '"'
      scope: punctuation.definition.string.begin.example-c
      push: double_quoted_string

    #
    - match: '//'
      scope: punctuation.definition.comment.example-c
      push: line_comment

    #                 keywoards
    - match: '\b(if|else|for|while|main|)\b'
```

```
    scope: keyword.control.example-c

  #
  - match: '\b(and|or|xor)\b'
    scope: keyword.operator.logical

  #
  - match: '\b(int|string|char|float|int|long)\b'
    scope: markup.raw.inline

  #                                TODO
  - match: '\b(TODO)\b'
    scope: markup.underline

  # Namespace
  - match: '\bstd\b'
    scope: constant.numeric.example-c

  #           .                                regexp
  - match: '\b(-)?[0-9.]+\b'
    scope: constant.numeric.example-c

double_quoted_string:
  - meta_scope: string.quoted.double.example-c
  - match: '\\.'
    scope: constant.character.escape.example-c
  - match: '"'
    scope: punctuation.definition.string.end.example-c
    pop: true

line_comment:
  - meta_scope: comment.line.example-c
  - match: $
    pop: true
```