Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление программная инженерия
Образовательная программа системное и прикладное программное обеспечение
Специализация системное программное обеспечение

ОТЧЕТ ПО ДОМАШНЕЙ РАБОТЕ № 2

курса «Разработка компайлеров»

по теме: «Регулярные грамматики»

Вариант № 9

Выполнил студент:
Тюрин Иван Николаевич
группа: P33102

Преподаватель:
Лаздин А. В.

Санкт-Петербург, 2024 г.

Содержание

Домашняя работа № 2
Регулярные грамматики

1. Задание варианта № 9

, , ,

По заданному регулярному выражению (см. вариант)

- Построить недетерминированный КА;
- По полученному НДА построить ДКА;
- Минимизировать полученный ДКА;
- Для минимального ДКА написать программу-распознаватель предложений языка, порождаемого регулярным выражением.

Продемонстрировать работу распознавателя на различных примерах (не менее трех правильных) предложений.

Использование символов + и ? в регулярных выражениях. Символ + используется для определения регулярного выражения, повторяющегося один или более раз. В этом смысле p+ = pp*. Символ ? используется для указания того, что регулярное выражение встречается ноль или один раз, тогда p? = ε | p.

, , ,

2. Выполнение задания

Заданеи варианта 9 можно видеть в таблице 1.1.

Вариант №	Регулярное выражение	Примечание
9	(a|b|c)*(a|b)*b	23

Таблица 1.1: Задание в соответствии с выданным вариантом

В соответствии с заданием построим граф состояний недетерминированного конечного автомата, соответствующего выданному регулярному выражению, см. рис. 1.1.

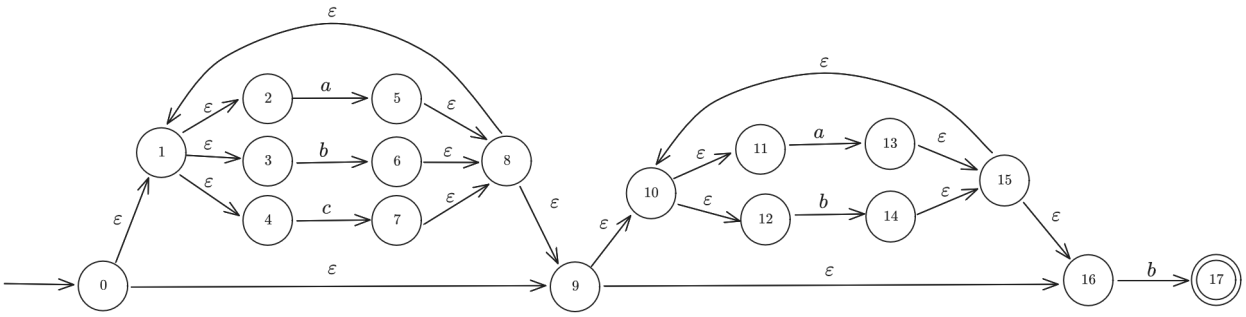

Рис. 1.1: Граф состояний и переходов между ними для недетерминированного конечного автомата

В соответствии с алгоритмом построения детерминированного конечного автомата из недетерминированного конечного автомата построим таблицу достижимых состояний 1.2.

Как можно видеть, всего получается 4 состояния; состояние с номером 2 является допускающим, т.к. из него достижимо допускающее состояние 17 изначального конечного автомата.

№	State	a	b	c
0	0, 1, 2, 3, 4,9, 10, 11, 12, 16	<1>	<2>	<3>
1	1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 15, 16	<1>	<2>	<3>
2	1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 14, 15, 16, 17	<1>	<2>	<3>
3	1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 16	<1>	<2>	<3>

Таблица 1.2: Состояния детерминированного конечного автомата достижимые в графе состояний недетерминированного конечного автомата

Теперь можно построить граф состояний и переходом между ними для детерминированного конечного автомата выполняющего разбор предложений регулярной грамматики, см. рис 1.2.

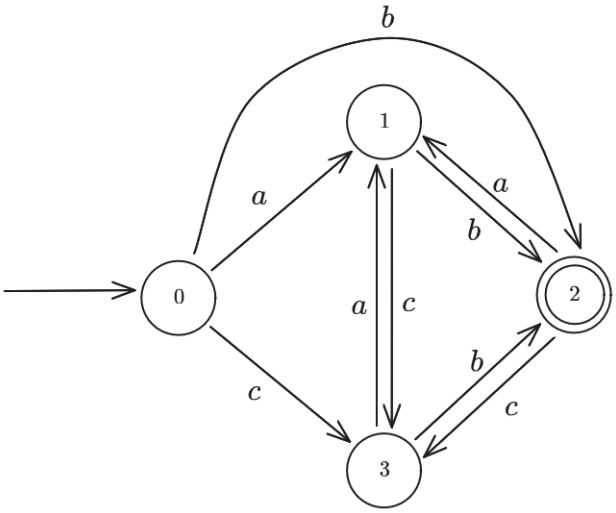

Рис. 1.2: Граф состояний и переходом между ними для детерминированного конечного автомата

Конечный автомат распознаватель был реализован на языке программирования Kotlin. Испытать его работу и посмотреть исходный код можно по ссылке https://pl.kotl.in/Cpg2pR63g, а так же в листинге 1.1.

```
class State<T> (
    val name: String,
    val reduce: (T) -> State<T>,
    val isAccepting: Boolean = false
) {
    override fun toString() = name
}

val basicReduce: (Char) -> State<Char> = {
    when(it) {
        'a' -> S1
        'b' -> S2
        'c' -> S3
        else -> SE
  }
}
val errorReduce: (Char) -> State<Char> = { SE }

val S0 = State<Char>("S0", basicReduce)
val S1 = State<Char>("S1", basicReduce)
val S2 = State<Char>("S2", basicReduce, isAccepting = true)
val S3 = State<Char>("S3", basicReduce)
val SE = State<Char>("Error", errorReduce)

object FSM {
    val initialState = S0
    val errorState = SE

    fun check(str: String, logger: ((Any) -> Unit)? = null): Boolean {
        var currState = initialState
        for (c in str) {
            logger?.invoke(currState)
            currState = currState.reduce(c)
            if (currState == errorState) break
```

```
        }
        logger?.invoke(currState)

        return currState.isAccepting
    }
}

fun main(args: Array<String>) {
    if (args.size < 1 || args.first().isBlank()) {
        println("Put testing strings as the program arguments")
    } else {
        for ((i, s) in args.withIndex()) {
            var trace = ""
            val result = FSM.check(s) {trace += ">$it"}
            println("${i + 1}) $s: $result ($trace)")
        }
    }
}
```

Листинг 1.1: Kotlin код реализующий требуемый конечный автомат

Пример работы программы можно видеть на рис. 1.3. В качестве аргументов командной строки на вход программе подаются строки, которые необходимо распознать.

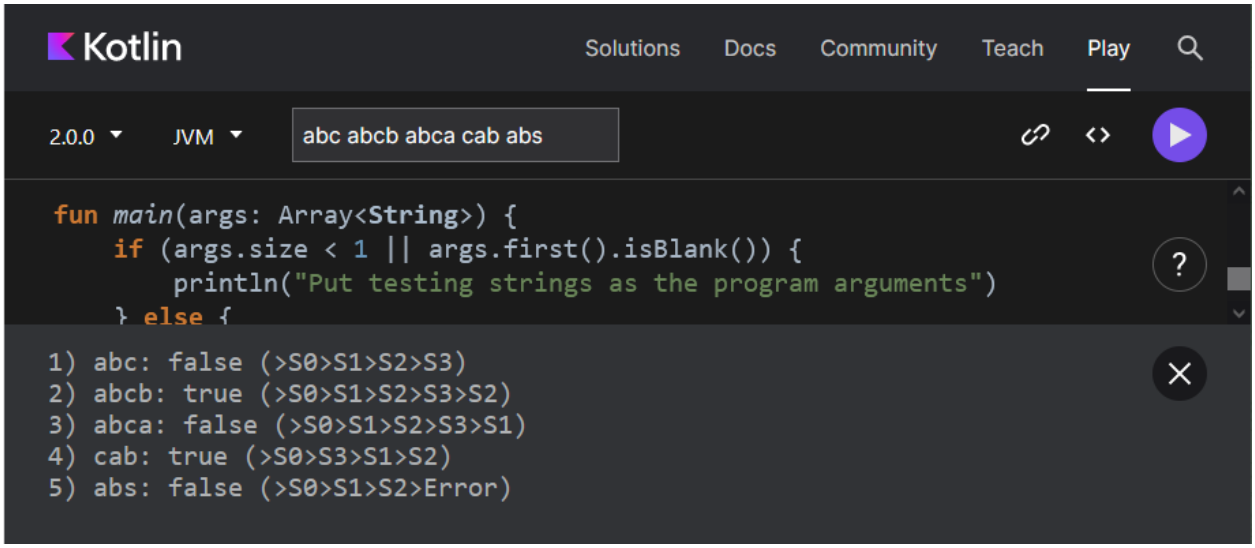

Рис. 1.3: Пример использования разработанной программы на языке программирования Kotlin на сайте play.kotlinlang.org

.

3. Вывод

В ходе выполнения работы был построен конечный автомат выполняющий распознавание символьных строк соответствующих заданному регулярному языку. Требуемый конечный автомат был реализован на языке программирования Kotlin.