ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Processamento de Linguagens

Licenciatura em Engenharia de Sistemas Informáticos

Trabalho Prático nº 1

Luís Gonçalves 18851 – a18851@alunos.ipca.pt

Luís Esteves 16960 – a16960@alunos.ipca.pt

Exercício I

Introdução

A análise léxica é um processo fundamental no desenvolvimento de compiladores e interpretadores para as linguagens de programação. A sua principal finalidade é transformar o código fonte numa sequência de tokens[1], ou símbolos, que representam as unidades léxicas da linguagem. Com esse intuito, utiliza-se uma abordagem baseada em expressões regulares, que são padrões de texto que descrevem as regras sintáticas da linguagem. Neste contexto, as linguagens regulares desempenham um papel supremo, dado que permitem especificar esses padrões de forma precisa e concisa.

Neste exercício, vamos explorar os conceitos básicos das linguagens regulares e como elas são utilizadas na análise léxica de linguagens de programação.

1- Um token é uma sequência de caracteres que representa uma unidade léxica da linguagem de programação em questão.

a) Definição da expressão regular

\(\-)? [0-9] +\. [0-9]+(e|E)(\-)?[0-9]+

```
# 1.A -> Definir expressão regular que reconheça constantes
com vírgula flutuante
# De salientar que no nosso projeto não aceitamos números
como por exemplo "+1.232e2"

expressão regular = "(\-)? [0-9] +\. [0-9] +(e|E)(\-)?[0-9]+"
```

b) Autómato Determinista

1. Esquema do Autómato e legenda correspondente

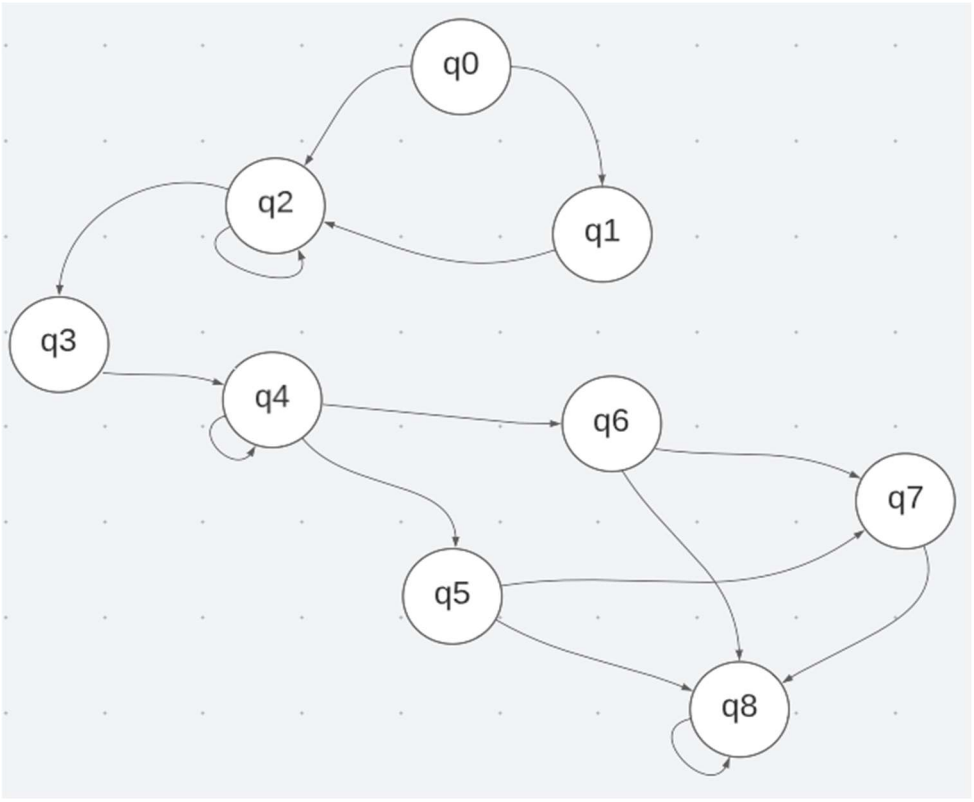


Figura 1- Autómato Finito Determinista

Legenda

q0	
q1	-
q2	[0-9]
q3	.
q4	[0-9]
q5	e
q6	E
q7	-
q8	[0-9]

2. Autómato Simplificado

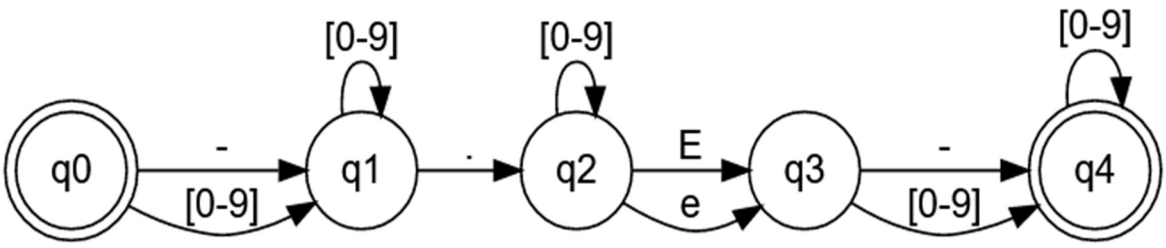


Figura 2- Autómato Finito Determinista Simplificado

3. Calcular o autómato determinista que implementa o reconhecimento da expressão regular em Python

```
#Em "Values" dizemos os valores que são admitidos pela expressão regular;
#Em "Q" são os estados no nosso autómato, agora já um autómato determinista

Values =
{"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "", "-", "e", "E", "\."}

Q = {"q0", "q1", "q2", "q3", "q4"}
```

c) Tabela de Transição e teste correspondente

1. Tabela de Transição

ESTADO	-	[0-9]	.	[0-9]	e	E	-	[0-9]
q0	q1	q2						
q1		q2						
q2		q2	q3					
q3				q4				
q4				q4	q5	q6		
q5							q7	q8
q6							q7	q8
q7								q8
q8								q8

2. Tabela de Transição Simplificada

ESTADO	ENTRADA	PRÓXIMO ESTADO
Q0	-	Q1
Q0	[0-9]	Q1
Q1	[0-9]	Q1
Q1	.	Q2
Q2	[0-9]	Q2
Q2	e	Q3
Q2	E	Q3
Q3	-	Q4
Q3	[0-9]	Q4
Q4	[0-9]	Q4

3. Definição da tabela de transição do autómato determinista e testar alguns exemplos em Python

```
#Demonstração da tabela de transição, replicada nesta lista de estados e
os valores que podem ser reconhecidos;


States = {
    "q0":{"-": "q1", "0": "q1", "1": "q1", "2": "q1", "3": "q1", "4":
"q1", "5": "q1", "6": "q1", "7": "q1",
          "8": "q1", "9": "q1"},
    "q1": {"0": "q1", "1": "q1", "2": "q1", "3": "q1", "4": "q1", "5":
"q1", "6": "q1", "7": "q1", "8": "q1",
          "9": "q1", ".": "q2"},
    "q2": {"0": "q2", "1": "q2", "2": "q2", "3": "q2", "4": "q2", "5":
"q2", "6": "q2", "7": "q2", "8": "q2", "9": "q2",
          "e": "q3", "E": "q3"},
    "q3": {"-": "q4", "0": "q4", "1": "q4", "2": "q4", "3": "q4", "4":
"q4", "5": "q4", "6": "q4", "7": "q4", "8": "q4",
          "9": "q4"},
    "q4": {"0": "q4", "1": "q4", "2": "q4", "3": "q4", "4": "q4", "5":
"q4", "6": "q4", "7": "q4", "8": "q4", "9": "q4",}
    }
```

4. Algoritmo

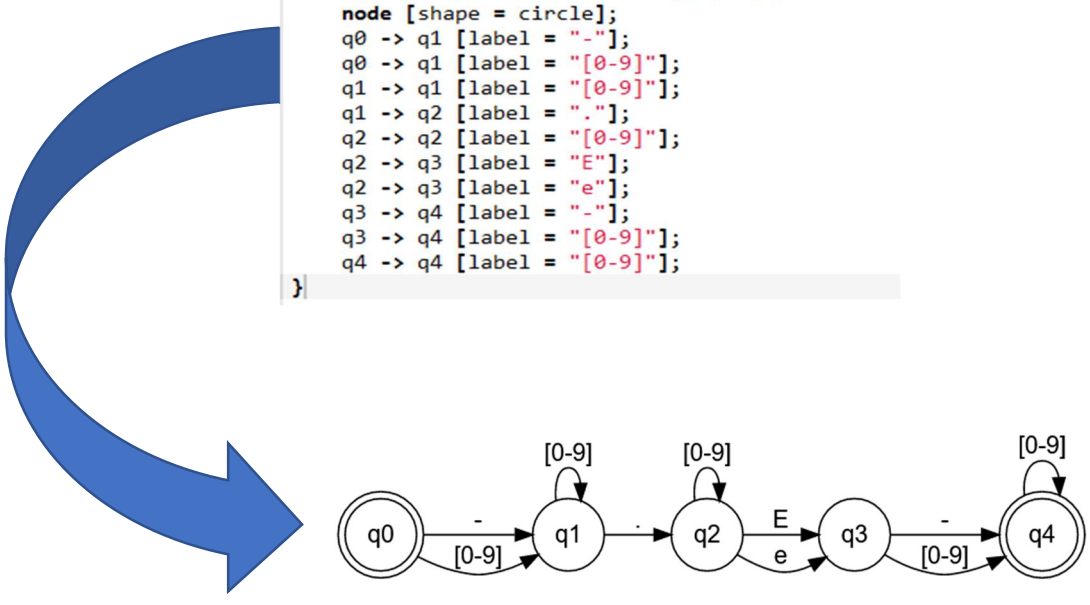
```
#Função que trata de reconhecer o input introduzido

inicial_State = "q0"
final_State = "q4"
def match(number):
    actual_State = inicial_State
    l = len(number)
    i = 0
    error = 0
    while (i < l) and (error != 1):
        actual_caracter = number[i]
        if (actual_caracter in States[actual_State]):
            actual_State =
States[actual_State][actual_caracter]
        else:
            error = 1
        i += 1
    return (actual_State in final_State) and (i == l)
```

Este algoritmo é responsável por reconhecer o input do user e verificar, a partir da tabela de transição definida em cima, se o input trata-se de uma constante com vírgula flutuante.

d) Grafo de saída

```
digraph representation_graph {
    fontname="Helvetica,Arial,sans-serif"
    node [fontname="Helvetica,Arial,sans-serif"]
    edge [fontname="Helvetica,Arial,sans-serif"]
    rankdir=LR;
    node [shape = doublecircle]; q0 q4;
    node [shape = circle];
    q0 -> q1 [label = "-"];
    q0 -> q1 [label = "[0-9]"];
    q1 -> q1 [label = "[0-9]"];
    q1 -> q2 [label = "."];
    q2 -> q2 [label = "[0-9]"];
    q2 -> q3 [label = "E"];
    q2 -> q3 [label = "e"];
    q3 -> q4 [label = "-"];
    q3 -> q4 [label = "[0-9]"];
    q4 -> q4 [label = "[0-9]"];
}
```



Bibliografia

https://www.ibm.com/docs/en/psfa/7.1.0?topic=constants-floating-point

http://magjac.com/graphviz-visual-editor/

https://regexr.com/

Pedro R. Santos, Thibault Langlois (2014) Compiladores – Da Teoria à Prática, FCA - Editora Informática