ESCOLA SUPERIOR DE TECNOLOGIA IPCA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Processamento de Linguagens

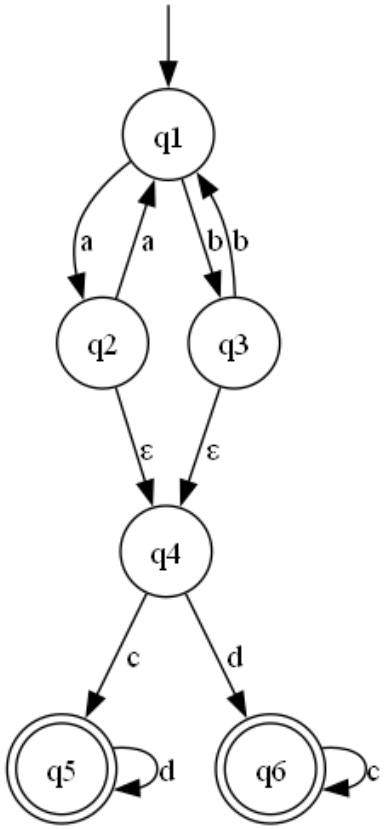
Engenharia de Sistemas Informáticos

Ano letivo de 2023/24

2. Expressões Regulares (ER) (cont.)

- construção de reconhecedores de ER com base em AFD

Autómato / Diagrama de grafos



Conjunto de estados: {q1,q2,q3,q4,q5,q6}

Estado inicial: q1

Estado final: {q5,q6}

Vocabulário: {a,b,c,d,ε}

Tabela de transições

δ	a	b	c	d	ε
q1	q2	q3			
q2	q1				q4
q3		q1			q4
q4			q5	q6	
q5					
q6					

Conjunto de estados: {q1,q2,q3,q4,q5,q6}

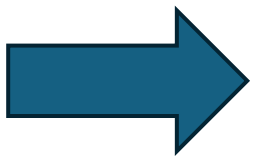
Estado inicial: q1

Estado final: {q5,q6}

Vocabulário: {a,b,c,d,ε}

Tabela de transições

δ	a	b	c	d	ε
q1	q2	q3			
q2	q1				q4
q3		q1			q4
q4			q5	q6	
q5					
q6					



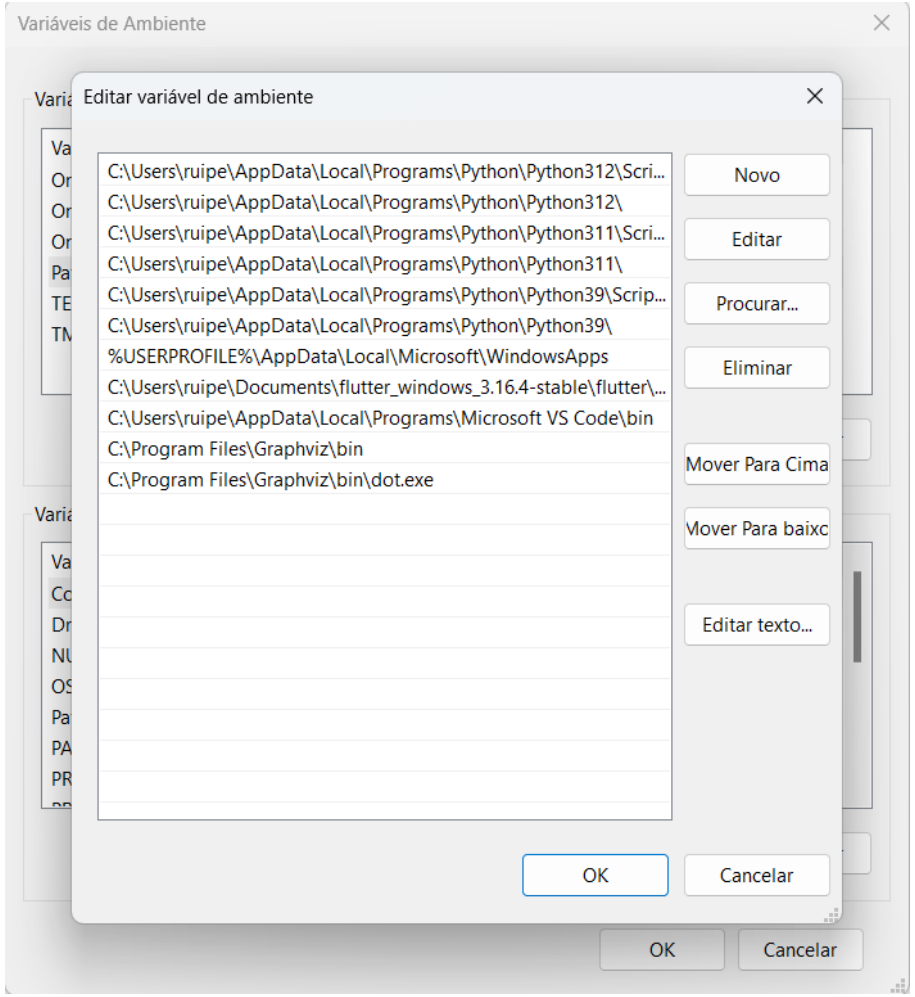
```
{
    "Q" : "",
    "V" : "",
    "q0" : "q1",
    "F" : ["q5","q6"],
    "delta" : {
        "q1" : {
            "a" : "q2",
            "b" : "q3"
        },
        "q2" : {
            "a" : "q1",
            "ε" : "q4"
        },
        "q3" : {
            "b" : "q1",
            "ε" : "q4"
        },
        "q4" : {
            "c" : "q5",
            "d" : "q6"
        },
        "q5" : {
            "d" : "q5"
        },
        "q6" : {
            "c" : "q6"
        }
    }
}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

1) Download graphviz https://graphviz.org/download/

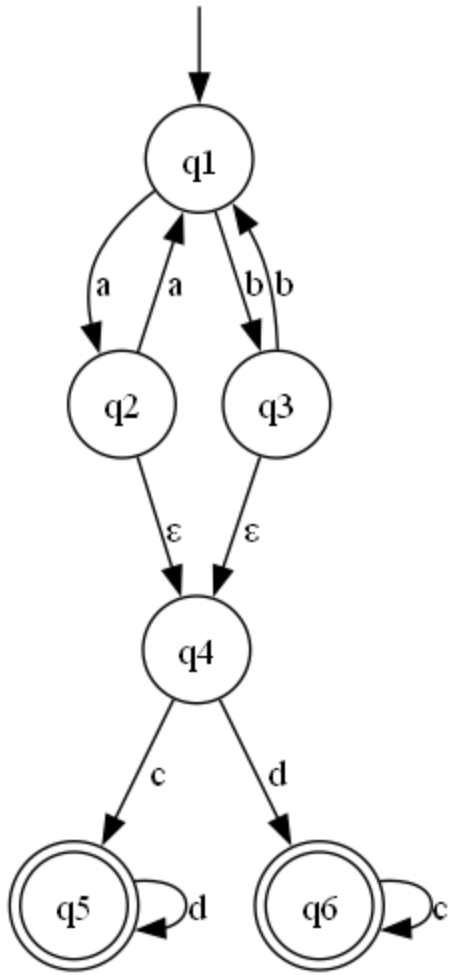
2) Adicionar o path do executável às variáveis de ambiente



ESCOLA SUPERIOR DE TECNOLOGIA IPCA

INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

```python
# Parte 1 - Gerar automato gráficamente usando a biblioteca graphviz

import json
from graphviz import Digraph

automato : dict = {}

with open("automato.json", "r", encoding="utf-8") as f:
    automato = json.load(f)

# Gerar o diagrama de grafos do automato
dot = Digraph(comment='Automato')

# 'none' para um node invisível
dot.node('start', shape='none', label='')
# criar transcição incial
dot.edge('start', automato["q0"], label='')

# Criar os estados
for state in automato["delta"].keys():

    # caso o estado seja final
    if state in automato["F"]:
        dot.node(state, state, shape="doublecircle")

    else:
        dot.node(state, state, shape="circle")

# Criar as transições
for estado_inicial, transitions in automato["delta"].items():
    for simbolo, estado_final in transitions.items():
        dot.edge(estado_inicial, estado_final, label = simbolo)

# Visualização
dot.render('automaton_graph', view=True, format='png')
```



```python
# Parte 2 - Gerar o código de funcionamento do automato
def reconhecedor(entrada : str, estado_inicial : str, transicoes : dict, estados_finais : list) -> bool:

    entrada : str = entrada.replace("ε", "")

    estado_atual : str = estado_inicial

    for char in entrada:
        # caso haja uma transição
        if char in transicoes[estado_atual]:
            estado_atual = transicoes[estado_atual][char]

        # caso haja uma transição com a palavra vazia
        elif "ε" in transicoes[estado_atual]:
            estado_atual_aux = transicoes[estado_atual]["ε"]
            if char in transicoes[estado_atual_aux]:
                estado_atual = transicoes[estado_atual_aux][char]
        else:
            # se não houver transição definida para este caracter de entrada, a palavra não é aceite
            return False
    return estado_atual in estados_finais

palavra : str = "aεεεεεcddddddddd"
estado_inicial : str = automato["q0"]
estados_finais : list = automato["F"]
transicoes : dict = automato["delta"]

reconhece : bool = reconhecedor(palavra, estado_inicial, transicoes, estados_finais)

if reconhece:
    print(f"A palavra '{palavra}' é aceite pelo automato.")
else:
    print(f"A palavra '{palavra}' não é aceite pelo automato.")
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

ArgParse

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

```python
## Exemplo de argparser
import argparse

def calculadora(args):
    if args.operator == 'soma':
        return args.number1 + args.number2
    elif args.operator == 'subtracao':
        return args.number1 - args.number2
    elif args.operator == 'multiplicacao':
        return args.number1 * args.number2
    elif args.operator == 'divisao':
        if args.number2 == 0:
            return "Erro: divisão por 0 não é válido."
        return args.number1 / args.number2
    else:
        return "Operador inválido."

def main():
    parser = argparse.ArgumentParser(description='Calculadora')
    parser.add_argument('operator', type=str, choices=['soma', 'subtracao', 'multiplicacao', 'divisao'],
                        help='Operações permitidas: soma, subtracao, multiplicacao, divisao')

    parser.add_argument('numero1', type=float, help='Primeiro numero')
    parser.add_argument('numero2', type=float, help='Segundo numero')

    args = parser.parse_args()
    result = calculadora(args)

    print(f"Result: {result}")

if __name__ == "__main__":
    main()
```

```
PS C:\Users\ruipe\OneDrive - Instituto Politécnico do Cávado e do Ave\Aulas\Processamento de Linguagens 2023 -2024\Código> python .\aula04_v2.py --help
usage: aula04_v2.py [-h] --operator {soma,subtracao,multiplicacao,divisao} --numero1 NUMERO1 --numero2 NUMERO2

Calculadora

options:
  -h, --help            show this help message and exit
  --operator {soma,subtracao,multiplicacao,divisao}
                        Operações permitidas: soma, subtracao, multiplicacao, divisao
  --numero1 NUMERO1     Primeiro numero
  --numero2 NUMERO2     Segundo numero
PS C:\Users\ruipe\OneDrive - Instituto Politécnico do Cávado e do Ave\Aulas\Processamento de Linguagens 2023 -2024\Código>
```

```
PS C:\Users\ruipe\OneDrive - Instituto Politécnico do Cávado e do Ave\Aulas\Processamento de Linguagens 2023 -2024\Código> python .\aula04_v2.py --operador soma --numero1 2 --numero2 3
Result: 5.0
PS C:\Users\ruipe\OneDrive - Instituto Politécnico do Cávado e do Ave\Aulas\Processamento de Linguagens 2023 -2024\Código>
```

Reconhecimento de uma palavra no AFD

Algoritmo de reconhecimento

```
Função reconhece(δ: TT, γ: T*): { aceite, erro }
    α ← q0
    Enquanto (γ ≠ ε) ∧ (α ≠ erro)
        α ← δ(α, head(γ))
        γ ← tail(γ)
    Se (α ∈ F) ∧ (γ = ε)
        r ← aceite
    Senão
        r ← erro
    Retornar r
```

Reconhecimento de uma palavra no AFD

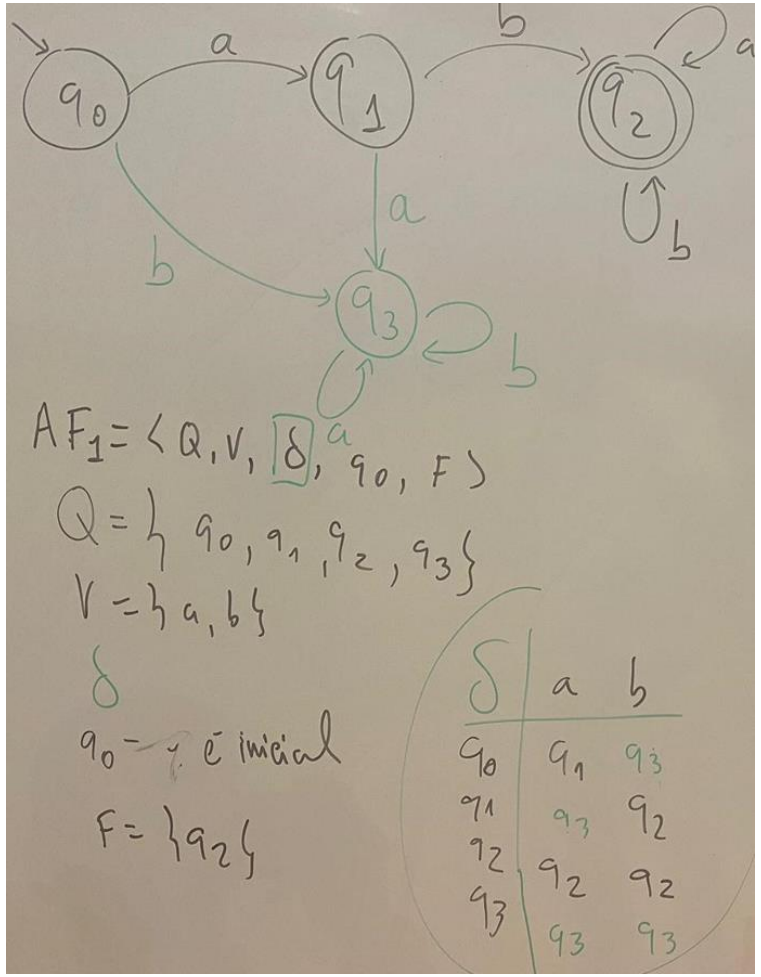
Implementação do algoritmo de reconhecimento em python

```
# definição do Autómato Finito
#    AF=(V,Q,delta,q0,F) tal que:
V = {"a", "b"}
Q = { "q0", "q1", "q2", "q3"}
delta = {   "q0": { "a":"q1", "b":"q3"},
            "q1": { "a":"q3", "b":"q2"},
            "q2": { "a":"q2", "b":"q2"},
            "q3": { "a":"q3", "b":"q3"},
            }
q0 = "q0"
F = {"q2"}
```

```
def reconhece(palavra: str) -> bool:
    estado_atual= q0
    tam = len(palavra)
    i = 0
    while (i<tam):
        simbolo_atual = palavra[i]
        estado_atual = delta[estado_atual][simbolo_atual]
        i+=1
    return (estado_atual in F)
```

Reconhecimento de uma palavra no AFD

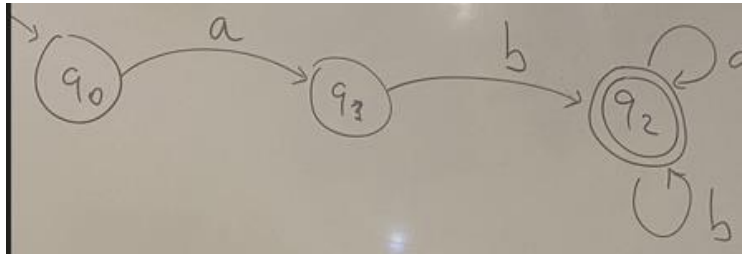
Implementação do algoritmo de reconhecimento em python



O **estado q3** está a funcionar como um estado de "erro"…

Qual o resultado de reconhece("abc") ?
'c' não faz parte do alfabeto, logo a palavra não é reconhecida

Segue a tabela de transição para o novo autómato, com o estado "erro" e a possibilidade da palavras de entrada ter outro símbolos



	a	b	outro
q_0	q_1	erro	erro
q_1	erro	q_2	erro
q_2	q_2	q_2	erro
outro	erro	erro	erro

Reconhecimento de uma palavra no AFD

Implementação do algoritmo de reconhecimento em Python

```
# definição do Autómato Finito
#    AF=(V,Q,delta,q0,F) tal que:
V = {"a", "b"}
Q = { "q0", "q1", "q2"}
delta = {"q0": { "a":"q1"},
         "q1": {"b":"q2"},
         "q2": {"a":"q2",
                "b":"q2"}}
q0 = "q0"
F = {"q2"}
```

```
def reconhece(palavra: str) -> bool:
    estado_atual= q0
    tam = len(palavra)
    i = 0
    while (i<tam) and (estado_atual != "erro"):
        simbolo_atual = palavra[i]
        if (simbolo_atual in delta[estado_atual]):
            estado_atual = delta[estado_atual][simbolo_atual]
        else:
            estado_atual = "erro"
        i+=1
    return (estado_atual in F)
```

	a	b
q_0	q_1	*n.d.*
q_1	*n.d.*	q_2
q_2	q_2	q_2

	a	b	*outro*
q_0	q_1	*erro*	erro
q_1	*erro*	q_2	erro
q_2	q_2	q_2	erro
outro	*erro*	*erro*	erro

Exemplos:

```
'a'   →  False
'aa'  →  False
'ab'  →  True
'aba' →  True
'abb' →  True
'ba'  →  False
'abc' →  False
'a.b' →  False
```