1º Exercício Programa – PCS3556-150-2019

Gustavo Ziyu Wang (8988790)

I. Introdução

Enunciado: *O objetivo do exercício é implementar o algoritmo de fecho reflexivo e transitivo de uma relação binária R ⊆ A × A sobre um conjunto finito A que é descrita por meio de um grafo direcionado. Conforme definido em sala de aula, a solução deverá construída por meio de recursão*

II. Implementação

1. Fecho Reflexivo: Em primeiro lugar, foi criada uma função para obter o fecho reflexivo (**reflexiveClosure**):

```
# REFLEXIVE_CLOSURE
# Retorna em closure os pares a serem adicionados para haver um fecho reflexivo na lista [h | t]

def reflexiveClosure(closure, [h | t]) do
    reflexiveClosure(join(closure, [{elem(h, 0), elem(h, 0)}, {elem(h, 1), elem(h, 1)}]), t)
end

def reflexiveClosure(closure, []) do
    closure
end
```

Foi necessária a criação de uma função **Join** para realizar a união de duas relações. Esta função deve evitar que haja dois pares iguais na relação.

```
# JOIN
# Esta função realiza a junção U de duas relações
# Verifica que a o par a ser adicionado não está presente na lista final

def join(list, [h | t]) do
    if h not in list do
        join(list ++ [h], t)
    else
        join(list, t)
    end
end

def join(list, []) do
    list
end
```

Os seguintes testes foram realizados para verificar que as funções funcionam corretamente:

1) União de duas relações com elementos distintos e semelhantes (join(lista1, lista2))

2) União com relação vazia (join(lista1, lista3))

3) União com a mesma lista (join(lista1, lista1))

```
iex(1)> lista1 = [{1, 2}, {2, 3}]
[{1, 2}, {2, 3}]
iex(2)> lista2 = [{2, 3}, {3, 4}]
[{2, 3}, {3, 4}]
iex(3)> lista3 = []
[]
```

```
iex(4)> Closure.join(lista1, lista2)
[{1, 2}, {2, 3}, {3, 4}]
iex(5)> Closure.join(lista1, lista3)
[{1, 2}, {2, 3}]
iex(6)> Closure.join(lista1, lista1)
[{1, 2}, {2, 3}]
```

Verifica-se que os resultados são exatamente o que era esperado.

Em seguida, foram realizados testes para a função reflexiveClosure:

1) Fecho reflexivo de uma relação reflexiva (reflexiveClosure([], lista1))

2) Fecho reflexivo de uma relação qualquer (reflexiveClosure([], lista2))

3) Fecho reflexivo de uma relação vazia (reflexiveClosure([], lista3))

```
iex(10)> lista1 = [{1, 1}, {2, 2}]
[{1, 1}, {2, 2}]
iex(11)> lista2 = [{1, 2}, {2, 3}, {5, 7}]
[{1, 2}, {2, 3}, {5, 7}]
iex(12)> lista3 = []
[]
```

```
iex(14)> Closure.reflexiveClosure([], lista1)
[{1, 1}, {2, 2}]
iex(15)> Closure.reflexiveClosure([], lista2)
[{1, 1}, {2, 2}, {3, 3}, {5, 5}, {7, 7}]
iex(16)> Closure.reflexiveClosure([], lista3)
[]
```

Assim como para a função Join, os testes obtiveram os resultados aguardados.

2. Fecho Transitivo: Com a função reflexiveClosure definida, o próximo passo é a implementação de uma função que realiza o fecho transitivo (**transitiveClosure**) de uma relação. O algoritmo usado é apresentado a seguir:

```
# TRANSITIVE_CLOSURE
# Retorna em closure o fecho transitivo da lista list

def transitiveClosure(closure, list, [h | t]) do
    transitiveClosure(join(closure, getTransitions(closure, h, list, list)), join(list, closure), t)
end

def transitiveClosure(closure, list, []) do
    join(list, closure)
end

def transitiveClosure([], list, [_h | _t]) do
    list
end
```

Ela também necessita de uma função auxiliar, para encontrar as transições dentro de uma relação. Esta função, **getTransitions**, é apresentada a seguir:

```
# GET_TRANSITIONS
# Retorna todas as transições

def getTransitions(transitions, {a, b}, [h | t], list) do
    cond do
        a == b -> getTransitions(transitions, {a, b}, t, list) # Caso seja uma relação reflexiva, vai para o
        b == elem(h, 0) -> getTransitions(join(transitions, [{a, elem(h, 1)}]), {a, elem(h, 1)}, list, list)
        a == elem(h, 1) -> getTransitions(join(transitions, [{elem(h, 0), b}]), {elem(h, 0), b}, list, list)
        true -> getTransitions(transitions, {a, b}, t, list) # Vai para o próximo elemento
    end
end

def getTransitions(transitions, {_a, _b}, [], _list) do
    transitions
end
```

Foram realizados os seguintes testes para verificar que a função obtém corretamente o fecho transitivo:

1) Fecho transitivo de uma relação qualquer (lista)

2) Fecho transitivo de uma relação vazia (lista2)

3) Fecho transitivo de uma relação sem fecho transitivo (lista3)

```
iex(3)> lista = [{1, 2}, {2, 3}, {2, 4}, {3, 5}, {2, 6}]
[{1, 2}, {2, 3}, {2, 4}, {3, 5}, {2, 6}]
iex(4)> lista2 = []
[]
iex(5)> lista3 = [{1, 2}, {3, 4}]
[{1, 2}, {3, 4}]
iex(6)> Closure.transitiveClosure([], lista, lista)
[{1, 2}, {2, 3}, {2, 4}, {3, 5}, {2, 6}, {1, 3}, {1, 5}, {1, 4}, {2, 5}, {1, 6}]
iex(7)> Closure.transitiveClosure([], lista2, lista2)
[]
iex(8)> Closure.transitiveClosure([], lista3, lista3)
[{1, 2}, {3, 4}]
```

3. Fecho Transitivo e Reflexivo: Com uma função que realiza o fecho transitivo e outra que realiza o fecho reflexivo, basta criar uma função adicional que efetua os dois fechos para obter o fecho transitivo e reflexivo.

```
# REFLEXIVE_TRANSITIVE_CLOSURE
# Exibe o fecho transitivo reflexivo da lista list

def reflexiveTransitiveClosure(list) do
    reflexiveClosure(join(list, transitiveClosure([], list, list)), list)
end
```

Foram realizados os seguintes testes para verificar o bom funcionamento da função:

1) Fecho de uma relação qualquer (lista)

2) Fecho de uma relação vazia (lista2)

3) Fecho de uma relação sem fecho transitivo (lista3)

3) Fecho de uma relação sem fecho reflexivo (lista4)

```
iex(9)> lista = [{1, 2}, {2, 3}, {2, 4}, {3, 5}, {2, 6}]
[{1, 2}, {2, 3}, {2, 4}, {3, 5}, {2, 6}]
iex(10)> lista2 = []
[]
iex(11)> lista3 = [{1, 3}, {4, 6}]
[{1, 3}, {4, 6}]
iex(12)> lista4 = [{1, 1}, {2, 2}]
[{1, 1}, {2, 2}]
iex(13)> Closure.reflexiveTransitiveClosure(lista)
[
  {1, 2},
  {2, 3},
  {2, 4},
  {3, 5},
  {2, 6},
  {1, 3},
  {1, 5},
  {1, 4},
  {2, 5},
  {1, 6},
  {1, 1},
  {2, 2},
  {3, 3},
  {4, 4},
  {5, 5},
  {6, 6}
]
iex(14)> Closure.reflexiveTransitiveClosure(lista2)
[]
iex(15)> Closure.reflexiveTransitiveClosure(lista3)
[{1, 3}, {4, 6}, {1, 1}, {3, 3}, {4, 4}, {6, 6}]
iex(16)> Closure.reflexiveTransitiveClosure(lista4)
[{1, 1}, {2, 2}]
```

III. Conclusão

Este primeiro exercício programa forneceu um primeiro contato com o paradigma de programação funcional. Ao contrário do paradigma imperativo, a linguagem Elixir é voltada à recursão e imutabilidade. Apesar das dificuldades encontradas ao migrar para esse novo paradigma, foi possível realizar o algoritmo de fecho transitivo e reflexivo nesta linguagem.