TP2 - Problema 2

November 15, 2022

1 TP2 - Problema 1

Grupo 11

Nelson Almeida a95652

Nuno Costa a97610 O Conway's Game of Life é um exemplo bastante conhecido de um autómato celular . Neste problema vamos modificar as regras do autómato da seguinte forma

a) O espaço de estados é finito definido por uma grelha de células booleanas (morta=0/viva=1) de dimensão $N \times N$ (com $N > 3$) identificadas por índices $(i, j) \in \{1..N\}$. Estas N^2 células são aqui referidas como "normais".

b) No estado inicial todas as células normais estão mortas excepto um quadrado 3×3, designado por "centro", aleatoriamente posicionado formado apenas por células vivas.

c) Adicionalmente existem $2\,N + 1$ "células da borda" que correspondem a um dos índices, i ou j, ser zero. As células da borda têm valores constantes que, no estado inicial, são gerados aleatoriamente com uma probabilidade ρ de estarem vivas.

d) As células normais o autómato modificam o estado de acordo com a regra "B3/S23": i.e. a célula nasce (passa de 0 a 1) se tem exatamente 3 vizinhos vivos e sobrevive (mantém-se viva) se o número de vizinhos vivos é 2 ou 3, caso contrário morre ou continua morta.

A célula (i_0, j_0) e (i_1, j_1) são vizinhas sse $(i_0 - i_1 = \pm 1) \vee (j_0 - j_1 = \pm 1)$

Pretende-se:

1. Construir uma máquina de estados finita que represente este autómato; são parâmetros do problema os parâmetros N, ρ e a posição do "centro".
2. Verificar se se conseguem provar as seguintes propriedades:
    1. Todos os estados acessíveis contém pelo menos uma célula viva.
    2. Toda a célula normal está viva pelo menos uma vez em algum estado acessível.

1.1 Inicialização

[49]:
```
from z3 import *
import random as rd
import numpy as np
import re
```

1.1.1 Declaração da matriz de estados

[50]:
```
def declare(k,N):
    state = {}
    for i in range(N):
        state[i]={}
        for j in range(N):
            state[i][j]={}
            state[i][j]["x"]=Int(f"frame_{k}_state_"+str(i)+","+str(j))
    return state
```

1.1.2 Inicialização dos valores de cada célula da matriz de acordo com as regras:

$for_{0<=i,j<N}((i,j) \in centro \implies estado[i][j] = 1) \vee (i = 0 \vee j = 0 \implies estado[i][j] = \rho * 1) \vee (estado[i][j] = 0)$

[51]:
```
def init(curr,N,centro, prob):
    instrucoes=[]
    centro = [(i,j) for i in range(centro[0],centro[0]+3) for j in␣
 ↪range(centro[1],centro[1]+3)]
    for i in range(N):
        for j in range(N):
            #celulas no centro
            if (i,j) in centro:
                instrucoes.append(curr[i][j]["x"]==1)
            elif i==0 or j==0:
                #celulas de borda
                num=rd.randint(0,100)
                instrucoes.append(curr[i][j]["x"]==1 if num<=prob*100 else␣
 ↪curr[i][j]["x"]==0)
            #todas as outras
            else:
                instrucoes.append(curr[i][j]["x"]==0)
    return And(instrucoes)
```

1.1.3 Definição da função de transição de acordo com as regras

Função que dados dois possíveis estados do programa, testa se é possível transitar de um estado para outro com base nas seguintes regras: Está morto e tem exatamente três vizinhos vivos=>vive

`Está vivo e tem exatamente dois ou três vizinhos vivos=>sobrevive`

`Está vivo e tem menos de dois vizinhos vivos ou mais de três vizinhos vivos=>morre`

`Está morto e tem menos de dois vizinhos vivos ou mais de três vizinhos vivos=>continua morto`

[52]:
```
def janelaVizinhos(x,y,N):
    minx=x-1
    maxx=x+1
```

```
    miny=y-1
    maxy=y+1
    if x-1<0:
        minx=0
    if x+1>=N:
        maxx=x
    if y-1<0:
        miny=0
    if y+1>=N:
        maxy=y
    return (minx,maxx,miny,maxy)


def trans(curr, prox,N):
    instrucoes=[]
    for i in range(N):
        for j in range(N):
            vizinhos=[]
            (minx,maxx,miny,maxy)=janelaVizinhos(i,j,N)
            for i1 in range(minx,maxx+1):
                for j1 in range(miny, maxy+1):
                    if (i1,j1)!=(i,j):
                        vizinhos.append(curr[i1][j1]["x"])
            #nasce
            t0=And(curr[i][j]["x"]==0,sum(vizinhos)==3, prox[i][j]["x"]==1)
            #sobrevive
            t1=And(curr[i][j]["x"]==1,Or(sum(vizinhos)==3,sum(vizinhos)==2),␣
 ↪prox[i][j]["x"]==1)
            #morre
            t2=And(curr[i][j]["x"]==1,Or(sum(vizinhos)>=4,sum(vizinhos)<=1),␣
 ↪prox[i][j]["x"]==0)
            #permanece morto
            t3=And(curr[i][j]["x"]==0,sum(vizinhos)!=3, prox[i][j]["x"]==0)
            instrucoes.append(Or(t0,t1,t2,t3))

    return And(instrucoes)
```

Função que dada uma função que gera uma cópia das variáveis do estado, um predicado que testa se um estado é inicial, um predicado que testa se um par de estados é uma transição válida, o valor das variáveis a, b e n e um número positivo k, para gerar um possível traço de execução do programa de tamanho k:

[88]:
```
def gera_traco(declare,init,trans,iters,probability,N, centro):
    if (N>=3):
        N=N+1
        s = Solver()
```

```
        trace = [declare(i,N) for i in range(iters)]

        s.add(init(trace[0],N,centro, probability))
        for i in range(iters-1):
            s.add(trans(trace[i],trace[i+1],N))

        if s.check() == sat:
            m = s.model()

            matrizes=[[[0 for k in range(N)] for j in range(N)] for i in␣
 ↪range(iters)]
            for itera in range(iters):
                for i in range(N):
                    for j in range(N):
                        matrizes[itera][i][j]=m[trace[itera][i][j]["x"]]

            for j in range(iters):
                print(f"Iteração {j}")
                for i in range(N):
                    print(matrizes[j][i])
                print("\n------------------\n")
        else:
            print("Não foi encontrada solução")
    else:
        print("N demasiado pequeno")
```

1.1.4 Exemplos

[54]:
```
N=4
centro=(1,1)
probability = rd.random()
gera_traco(declare,init,trans,6,probability,N,centro)
```

```
Iteração 0
[0, 0, 0, 0, 1]
[0, 1, 1, 1, 0]
[1, 1, 1, 1, 0]
[0, 1, 1, 1, 0]
[1, 0, 0, 0, 0]

------------------

Iteração 1
[0, 0, 1, 1, 0]
[1, 0, 0, 0, 1]
[1, 0, 0, 0, 1]
[0, 0, 0, 1, 0]
```

```
[0, 1, 1, 0, 0]

------------------

Iteração 2
[0, 0, 0, 1, 0]
[0, 1, 0, 0, 1]
[0, 0, 0, 1, 1]
[0, 1, 1, 1, 0]
[0, 0, 1, 0, 0]

------------------

Iteração 3
[0, 0, 0, 0, 0]
[0, 0, 1, 0, 1]
[0, 1, 0, 0, 1]
[0, 1, 0, 0, 1]
[0, 1, 1, 1, 0]

------------------

Iteração 4
[0, 0, 0, 0, 0]
[0, 0, 0, 1, 0]
[0, 1, 1, 0, 1]
[1, 1, 0, 0, 1]
[0, 1, 1, 1, 0]

------------------

Iteração 5
[0, 0, 0, 0, 0]
[0, 0, 1, 1, 0]
[1, 1, 1, 0, 1]
[1, 0, 0, 0, 1]
[1, 1, 1, 1, 0]

------------------
```

[55]:
```
N=1
centro=(1,1)
probability = rd.random()
gera_traco(declare,init,trans,6,probability,N,centro)
```

```
N demasiado pequeno
```

[56]:
```
N=20
centro=(50,20)
probability=rd.random()
gera_traco(declare,init,trans,3,probability,N,centro)
```

```
Iteração 0
[0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

------------------

Iteração 1
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

------------------

Iteração 2
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

------------------
```

1.1.5 Função que verifica se para todos os estados existe pelo menos uma célula viva

Esta função verifica se para todos os estados existe pelo menos uma célula vazia através da criação de uma função auxiliar, neste caso de nome invariante que recebe a matriz de estados a cada iteração da atualização da matriz e retorna a regra lógica de que a soma do estado de todas as células da matriz tem de ser maior do que zero, que será adicionada ao solver como restrição.

[57]:
```python
def b1(declare,init,trans,iters,probability,N, centro,invariante):
    if (N>=3):
        s = Solver()
        trace = [declare(i,N) for i in range(iters)]

        s.add(init(trace[0],N, centro, probability))
        s.add(invariante(trace[0],N))
```

```
        for i in range(iters-1):
            s.add(trans(trace[i],trace[i+1],N))
            s.add(invariante(trace[i+1],N))


        if s.check() == sat:
            m = s.model()
            print(f"Propriedade válida nos primeiros {iters} passos")
            print("\n------------------\n")
            matrizes=[[[0 for k in range(N)] for j in range(N)] for i in␣
 ↪range(iters)]
            for itera in range(iters):
                for i in range(N):
                    for j in range(N):
                        matrizes[itera][i][j]=m[trace[itera][i][j]["x"]]
            for j in range(iters):
                print(f"Iteração {j}")
                for i in range(N):
                    print(matrizes[j][i])
                print("\n------------------\n")
        else:
            print("Propriedade inválida")

def invariante(curr,N):
    values=[]
    for i in range(N):
        for j in range(N):
            values.append(curr[i][j]["x"])

    return sum(values)>0
```

1.1.6 Exemplos

[61]:
```
N=4
centro=(1,1)
probability=rd.random()
b1(declare,init,trans,6,probability,N,centro,invariante)
```

```
Propriedade válida nos primeiros 6 passos

------------------

Iteração 0
[0, 0, 0, 0]
[0, 1, 1, 1]
[0, 1, 1, 1]
```

```
[0, 1, 1, 1]

------------------

Iteração 1
[0, 0, 1, 0]
[0, 1, 0, 1]
[1, 0, 0, 0]
[0, 1, 0, 1]

------------------

Iteração 2
[0, 0, 1, 0]
[0, 1, 1, 0]
[1, 1, 0, 0]
[0, 0, 0, 0]

------------------

Iteração 3
[0, 1, 1, 0]
[1, 0, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

------------------

Iteração 4
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 1, 0]
[0, 1, 0, 0]

------------------

Iteração 5
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 1, 0]
[0, 1, 0, 0]

------------------
```

[65]:
```
N=4
centro=(1,1)
```

```
probability=1
b1(declare,init,trans,6,probability,N,centro,invariante)
```

```
Propriedade inválida
```

1.1.7 Função que verifica se toda a célula normal está viva pelo menos uma vez em algum estado acessível

Esta função verifica se toda a célula normal está viva pelo menos uma vez em algum estado acessível criando uma matriz auxiliar que é transita para de estado para estado com as seguintes regras:

```
-o estado atual da matriz é vivo e o estado da matriz auxiliar nessa célula está marcado como r

-o estado atual da matriz é vivo e o estado da matriz auxiliar nessa célula está marcado como v

-o estado atual da matriz é morto e o estado da matriz auxiliar nessa célula está marcado como

-o estado atual da matriz é morto e o estado da matriz auxiliar nessa célula está marcado como
```

Na última iteração da matriz, teríamos na matriz auxiliar todas as células que ao longo da evolução da matriz tiveram estados vivos, o somatório dessa matriz com exceção das células de borda teria de ser igual ao n*n, o que verificaria a propriedade que queremos demonstrar

[93]:
```
def declareInvariante(k,N):
    state = {}
    for i in range(N):
        state[i]={}
        for j in range(N):
            state[i][j]={}
            ␣
 ↪state[i][j]["x"]=Int(f"frameInvariante_{k}_state_"+str(i-1)+","+str(j-1))
    return state

def initInvariante(curr,N):
    instrucoes=[]
    for i in range(N):
        for j in range(N):
            instrucoes.append(curr[i][j]["x"]==0)
    return And(instrucoes)

def transInvariante(currInvariante, proxInvariante, currState,N):
    instrucoes=[]
    for i in range(N):
        for j in range(N):
            t0=And(currInvariante[i][j]["x"]==0, currState[i][j]["x"]==0,␣
 ↪proxInvariante[i][j]["x"]==0)
            t1=And(currInvariante[i][j]["x"]==0, currState[i][j]["x"]==1,␣
 ↪proxInvariante[i][j]["x"]==1)
```

```
            t2=And(currInvariante[i][j]["x"]==1, currState[i][j]["x"]==0,␣
 ↪proxInvariante[i][j]["x"]==1)
            t3=And(currInvariante[i][j]["x"]==1, currState[i][j]["x"]==1,␣
 ↪proxInvariante[i][j]["x"]==1)
            instrucoes.append(Or(t0,t1,t2,t3))
    return And(instrucoes)


def␣
 ↪b2(declare,init,trans,iters,probability,N,centro,declareInvariante,initInvariante,transInva
 ↪
    if (N>=3):
        N=N+1
        s = Solver()
        trace = [declare(i,N) for i in range(iters)]
        traceInvariante = [declareInvariante(i,N) for i in range(iters)]

        s.add(init(trace[0],N,centro,probability))
        s.add(initInvariante(traceInvariante[0],N))

        for i in range(iters-1):
            s.add(trans(trace[i],trace[i+1],N))
            s.
 ↪add(transInvariante(traceInvariante[i],traceInvariante[i+1],trace[i], N))


        if s.check() == sat:
            m = s.model()
            aliveAtLeastOnce=0
            for i in range(1,N):
                for j in range(1,N):
                    aliveAtLeastOnce+=m[traceInvariante[iters-1][i][j]["x"]].
 ↪as_long()
            if (aliveAtLeastOnce==((N-1)*(N-1))):
                print("A propriedade é verdadeira")
            else:
                print("A propriedade é falsa")

            ############
            matrizes=[[[0 for k in range(N)] for j in range(N)] for i in␣
 ↪range(iters)]
            for itera in range(iters):
                for i in range(N):
                    for j in range(N):
                        matrizes[itera][i][j]=m[trace[itera][i][j]["x"]]
```

```
            for j in range(iters):
                print(f"Iteração {j}")
                for i in range(N):
                    print(matrizes[j][i])
                print("\n------------------\n")


            #########
            matrizes=[[[0 for k in range(N)] for j in range(N)] for i in␣
 ↪range(iters)]
            for itera in range(iters):
                for i in range(N):
                    for j in range(N):
                        ␣
 ↪matrizes[itera][i][j]=m[traceInvariante[itera][i][j]["x"]]

            for j in range(iters):
                print(f"IteraçãoInvariante {j}")
                for i in range(N):
                    print(matrizes[j][i])
                print("\n------------------\n")
    else:
        print("N demasiado pequeno")
```

1.1.8 Exemplos

[94]:
```
N=4
centro=(1,1)
probability=rd.random()
b2(declare,init,trans,6,probability,N,centro, declareInvariante,␣
 ↪initInvariante, transInvariante)
```

```
A propriedade é falsa
Iteração 0
[0, 0, 0, 0, 0]
[0, 1, 1, 1, 0]
[0, 1, 1, 1, 0]
[0, 1, 1, 1, 0]
[0, 0, 0, 0, 0]

------------------

Iteração 1
[0, 0, 1, 0, 0]
[0, 1, 0, 1, 0]
[1, 0, 0, 0, 1]
```

```
[0, 1, 0, 1, 0]
[0, 0, 1, 0, 0]

------------------

Iteração 2
[0, 0, 1, 0, 0]
[0, 1, 1, 1, 0]
[1, 1, 0, 1, 1]
[0, 1, 1, 1, 0]
[0, 0, 1, 0, 0]

------------------

Iteração 3
[0, 1, 1, 1, 0]
[1, 0, 0, 0, 1]
[1, 0, 0, 0, 1]
[1, 0, 0, 0, 1]
[0, 1, 1, 1, 0]

------------------

Iteração 4
[0, 1, 1, 1, 0]
[1, 0, 1, 0, 1]
[1, 1, 0, 1, 1]
[1, 0, 1, 0, 1]
[0, 1, 1, 1, 0]

------------------

Iteração 5
[0, 1, 1, 1, 0]
[1, 0, 0, 0, 1]
[1, 0, 0, 0, 1]
[1, 0, 0, 0, 1]
[0, 1, 1, 1, 0]

------------------

IteraçãoInvariante 0
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
```

```
------------------

IteraçãoInvariante 1
[0, 0, 0, 0, 0]
[0, 1, 1, 1, 0]
[0, 1, 1, 1, 0]
[0, 1, 1, 1, 0]
[0, 0, 0, 0, 0]

------------------

IteraçãoInvariante 2
[0, 0, 1, 0, 0]
[0, 1, 1, 1, 0]
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 0]
[0, 0, 1, 0, 0]

------------------

IteraçãoInvariante 3
[0, 0, 1, 0, 0]
[0, 1, 1, 1, 0]
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 0]
[0, 0, 1, 0, 0]

------------------

IteraçãoInvariante 4
[0, 1, 1, 1, 0]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 0]

------------------

IteraçãoInvariante 5
[0, 1, 1, 1, 0]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[1, 1, 1, 1, 1]
[0, 1, 1, 1, 0]

------------------
```

[95]:
```
N=3
centro=(1,1)
probability=rd.random()
b2(declare,init,trans,6,probability,N,centro, declareInvariante,␣
  ↪initInvariante, transInvariante)
```

```
A propriedade é verdadeira
Iteração 0
[0, 0, 0, 0]
[0, 1, 1, 1]
[0, 1, 1, 1]
[0, 1, 1, 1]

------------------

Iteração 1
[0, 0, 1, 0]
[0, 1, 0, 1]
[1, 0, 0, 0]
[0, 1, 0, 1]

------------------

Iteração 2
[0, 0, 1, 0]
[0, 1, 1, 0]
[1, 1, 0, 0]
[0, 0, 0, 0]

------------------

Iteração 3
[0, 1, 1, 0]
[1, 0, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]

------------------

Iteração 4
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 1, 0]
[0, 1, 0, 0]

------------------
```

```
Iteração 5
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 0, 1, 0]
[0, 1, 0, 0]

------------------

IteraçãoInvariante 0
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]

------------------

IteraçãoInvariante 1
[0, 0, 0, 0]
[0, 1, 1, 1]
[0, 1, 1, 1]
[0, 1, 1, 1]

------------------

IteraçãoInvariante 2
[0, 0, 1, 0]
[0, 1, 1, 1]
[1, 1, 1, 1]
[0, 1, 1, 1]

------------------

IteraçãoInvariante 3
[0, 0, 1, 0]
[0, 1, 1, 1]
[1, 1, 1, 1]
[0, 1, 1, 1]

------------------

IteraçãoInvariante 4
[0, 1, 1, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
[0, 1, 1, 1]

------------------
```

```
IteraçãoInvariante 5
[0, 1, 1, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
[0, 1, 1, 1]

------------------
```

[96]:
```
N=2
centro=(1,1)
probability=rd.random()
b2(declare,init,trans,6,probability,N,centro, declareInvariante,␣
  ↪initInvariante, transInvariante)
```

```
N demasiado pequeno
```