ISM MP, Informatique
Année 2024/2025

Corrigé du TP Informatique 13

Exercice 1

On saisit :

```
def transpose(G):
    """transpose(G:dict)->dict
    Calcule le graphe transposé de G décrit par liste d'adjacence"""
    tG={s:[] for s in G}
    for v in G:
        for s in G[v]:
            tG[s].append(v)
    return tG
```

Exercice 2

1. On saisit :

```
def attracteur(G,S_i,U):
    """attracteur(G:dict,S_i:dict,U:dict)->dict
    Calcul de l'attracteur Attr^i(U)
    G : graphe décrit par liste d'adjacence
    S_i : sommets contrôlé le joueur J_i
    U : objectif"""
    A=[]
    nG={}
    for v in G:
        nG[v]=len(G[v])
    tG=transpose(G)
    def parcours(u):
        if u not in A:
            A.append(u)
            for v in tG[u]:
                nG[v]=nG[v]-1
                if v in S_i or nG[v]==0:
                    parcours(v)
    for u in U:
        parcours(u)
    return A
```

2. Pour le jeu n°1, on saisit :

```
G={1:[4,5,6],2:[5,6],3:[],4:[2,3],5:[3],6:[]}
S_0=[1,2,3]
V_0=[6]
S_1=[4,5,6]
V_1=[3]
```

On obtient :

```
Jeu n°1
Attr^0(V_0)= [6, 1, 2]
Attr^1(V_1)= [3, 4, 5]
```

On observe $\mathrm{Attr}^0(\mathrm{V}_0) \sqcup \mathrm{Attr}^1(\mathrm{V}_1) = \mathrm{S}$ ce qui est conforme à la théorie : un jeu d'accessibilité à deux joueurs sans match nul (ni cycle, ni cul-de-sac) est déterminé.

Pour le jeu n°2, on saisit :

```
G={1:[4,5,6],2:[5,6],3:[],4:[3],5:[2],6:[]}
S_0=[1,2,3]
V_0=[6]
S_1=[4,5,6]
V_1=[3]
```

On obtient :

```
Jeu n°2
Attr^0(V_0)= [6, 1, 2, 5]
Attr^1(V_1)= [3, 4]
```

Malgré la présence évidente d'un cycle, on observe $\mathrm{Attr}^0(\mathrm{V}_0) \sqcup \mathrm{Attr}^1(\mathrm{V}_1) = \mathrm{S}$. Depuis le sommet 2, le joueur J_0 dispose d'une stratégie gagnante et comme il joue rationnellement, il ne s'engage pas dans un cycle qui vaut comme match nul. Ainsi, même en présence d'un cycle, un jeu d'accessibilité peut être déterminé.

Pour le jeu n°3, on saisit :

```
G={1:[5,6,7,8],2:[6],3:[8],4:[],5:[3,4],6:[2],7:[],8:[]}
S_0=[1,2,3,4]
V_0=[8]
S_1=[5,6,7,8]
V_1=[4]
```

On obtient :

```
Jeu n°3
Attr^0(V_0)= [8, 1, 3]
Attr^1(V_1)= [4, 5]
```

On observe $\mathrm{Attr}^0(\mathrm{V}_0) \sqcup \mathrm{Attr}^1(\mathrm{V}_1) \subsetneq \mathrm{S}$. Ceci s'explique par la présence du sommet 7 qui est un cul-de-sac et par la possibilité de parties infinies avec le cycle $2 - 6$ qui sont inévitables si on atteint 6.

Exercice 3

On modifie légèrement la fonction de calcul d'un attracteur pour construire un « squelette » de stratégie depuis tous les sommets du joueur J_i concerné qui sont dans son attracteur. Il reste ensuite à finaliser la construction des stratégies en les définissant de manière quelconque sur les sommets contrôlés par J_i mais qui ne sont pas dans son attracteur. On écrit la fonction `complete_strat` qui complète le squelette de stratégie avec notamment une fonction récursive locale qui réalise la complétion pour tous les sommets contrôlés par J_i, hors de l'attracteur et qui ne sont pas des états finals.

```
def attracteur_strat(G,S_i,U):
    A=[]
    nG={}
    strategie={}
    for v in G:
        nG[v]=len(G[v])
    tG=transpose(G)
    def parcours(u):
        if u not in A:
            A.append(u)
            for v in tG[u]:
                nG[v]=nG[v]-1
                if v in S_i or nG[v]==0:
                    parcours(v)
                    if v in Si:
                        strategie[v]=u
    for u in U:
        parcours(u)
    return A,strategie


def complete_strat(G,S_i,strategie):
    absents=[]
    for s in S_i:
        if s not in strategie and len(G[s])>0:
            absents.append(s)
    liste_strat=[strategie]
    def comp_rec(absents,liste_strat):
        if len(absents)==0:
            return liste_strat
        s=absents.pop()
        aux=[]
        for succ in G[s]:
            for strat in liste_strat:
                dico=dict(strat)
                dico[s]=succ
                aux.append(dico)
        return comp_rec(absents,aux)
    return comp_rec(absents,liste_strat)
```

On teste pour le jeu de Nim :

```
arene = {(1 ,0) :[], (1 ,1) :[],
         (2 ,0) :[(1 ,1)], (2 ,1) :[(1 ,0)],
         (3,0):[(2,1),(1,1)],(3,1):[(2,0),(1,0)]}
for k in range (4 ,10):
    for i in [0,1]: # J_0 et J_1
        arene [(k,i) ]=[(k -3,1-i),(k -2,1-i),(k-1,1-i)]

S_0=[(k,0) for k in range(1,10)]
V_0=[(1,1)]

print("Sommets contrôlés par J_0 dans Attr^0(V_0)")
print()
att_0,strat_0=attracteur_strat(arene,S_0,V_0)
print(list(strat_0))
print()

print("Construction des stratégies")
print()
liste_strat=complete_strat(arene,S_0,strat_0)
for strat in liste_strat:
    print(strat)
    print()
```

On obtient :

```
Sommets contrôlés par J_0 dans Attr^0(V_0)

[(2, 0), (3, 0), (6, 0), (7, 0), (8, 0), (4, 0)]

Construction des stratégies

{(2, 0): (1, 1), (3, 0): (1, 1), (6, 0): (5, 1), (7, 0): (5, 1),
 (8, 0): (5, 1), (4, 0): (1, 1), (9, 0): (6, 1), (5, 0): (2, 1)}

{(2, 0): (1, 1), (3, 0): (1, 1), (6, 0): (5, 1), (7, 0): (5, 1),
 (8, 0): (5, 1), (4, 0): (1, 1), (9, 0): (7, 1), (5, 0): (2, 1)}

...
```

avec un total de 9 stratégies en tout, gagnantes pour J_0 depuis les sommets 8, 7, 6, 4, 3, 2.