DLST L1 — Université Grenoble Alpes

Travaux Pratiques — INF202 2018/2019

Feuille 1

Note Ce TP sera à réalisé en Python sur deux documents séparés :

anneau01.py

relation.py

Le premier document contiendra les fonctions que vous allez pouvoir réutiliser dans les séances suivantes, ces fonctions seront codées pour être utilisées sur différents anneaux : booléens , anneau min-plus, anneau entier classique, anneau division, réels.
Vous remarquerez que ces fonctions prendront en argument certaine parmi *unit(), zero(), addition(), multiplication().*

Le second document est l'ensemble des fonctions spécifique de la séance et commencera par importer le premier.

Prérequis

Si on considère deux ensembles A et B, une relation binaire entre A et B est une liste d'association entre des éléments de A et des éléments de B. Elle peut être également vue comme une fonction de $A \times B$ *dans l'ensemble* $\{0, 1\}$*, retournant vrai sur* (a, b) *si* (a, b) *appartient à la relation.*

Booléens

L'anneau des booléens (Boolean ring) en anglais est : $\{\{vrai, faux\}, faux, ou, vrai, et\}$*, également décrit sous la forme* $\{\{0, 1\}, 0, +, 1, .\}$*, On utilisera la seconde forme.*

Dans la suite, on utilisera les fonctions *unit() zero()*, *produit(a,b)* et *somme(a,b)* que l'on codera dans anneau01.py.

```
def unit():    return 1

  def produit(a,b):
 if (a==1 and b==1)
   return 1
 else
   return 0
```

coder similairement *zero* et *somme*, qui retournent la constante 0 (élément absorbant des booléens) et qui code le *ou* logique.

Affichage

Dans ce TP, nous codons les relation binaire par des tableaux en deux dimension de booléens : $aRb \Leftrightarrow R[a][b] \neq 0$.
Les tableaux sont au choix des listes de listes ou ceux construit par la bibliothèque Numpy, similaires à ceux du cours.

Écrire une fonction *affiche(R)* dans relation.py.
Affiche dessine (avec print) à l'écran un tableau en deux dimensions de taille $b1 \times b2$, avec le contenu de R.
On affichera un nombre de lignes égal à la taille de R, et un nombre de colonnes égal à la taille de R[0] ou R[i].

Tester *affiche([[0,1,0,1,0][0,0,0,0,0][1,0,0,0,1][0,1,1,1,0]],4,5)*

Relation simples

Un prédicat est une fonction qui renvoie un booléen. Un exemple est le prédicat vide :
> *def vide(a,b) :*
> *return False*

Dans relation.py,

- construire le prédicat *successeur* : $iRj \Leftrightarrow i = j + 1$,
- construire le prédicat *moitié* : $iRj \Leftrightarrow i = \lfloor \frac{j}{2} \rfloor$,
- construire le prédicat *diviseur non trivial* : $iRj \Leftrightarrow i|j$ et $i \notin \{1, j\}$,
- écrire une fonction *conversion(p,a,b)* qui retourne la relation définie sur $[1, a] \times [1, b]$ à partir du prédicat p.

Afficher (avec la fonction affiche()) les relations *successeurs*, *moitié* et *diviseur*.

Opérations sur les relations

Exemple : Symétrie *La relation symétrique d'une relation sur deux ensembles A et B est la relation sur B et A avec les mêmes associations.* $\forall i, j,\ iRj \Leftrightarrow jSi$

```
def symetrie(R):
  S= [ ]
  for i in range(0,len(R))
     ligne=[ ]
     for j in range(0,len(R[0]))
        ligne.append(R[ j ][ i ])
     S.append(ligne)
  return S
```

Utiliser symetrie() sur *successeur* et afficher (avec *affiche()*) le résultat.

Composition *Si on considère trois ensembles A,B et C, et deux relations R sur* $A \times B$ *et S sur* $B \times C$*, la composition des relations binaires S et R est une relation sur* $A \times C$ *notée S o R (par les mathématiciens) et R ; S (par les informaticiens) et définie par a SoR c si et seulement si il existe b dans B tel que aRb et bSc.*
C'est un cas particulier de produit de matrices R et S sur l'anneau des booléens

Écrire dans relation.py une fonction *composition(R,S,et,ou)* retournant la composée RoS.
Les arguments sont les deux relation, quelle fonction ou utiliser (lors d'un appel, ce sera somme) et quelle fonction et utiliser (produit).
On peut s'inspirer du produit matriciel en se rappelant que $xRoSy$ signifie $\exists z, xz \in S$ et $zy \in R$ et peut s'écrire
$(xSz_1$ et $z_1Ry)$ ou $(xSz_2$ et $z_2Ry)$ ou $(xSz_3$ et $z_3Ry)$ ou ...

Utiliser cette fonction sur *successeur* appliquée à elle même et appliquer affiche() au résultat.

Union *L'union de deux relations sur les mêmes ensembles est l'union des listes d'associations. id est :* $aR \cup Sb$ *si et seulement si aRb ou aSb*

Écrire dans relation.py une fonction *union(R,S,ou)* qui renvoie l'union.
On se rappelle que $xRuSy$ peut s'écrire xRy ou xSy

Utiliser cette fonction entre *successeur* et *prédécesseur*. Afficher le résultat.

On note S^1 la relation *successeur*, $S^2 = S^1oS^1$ et $S^n = S^1oS^{n-1}$. Afficher $S^1 \cup S^2$ (en python, notez simplement S1 et S2)

Intersection *L'union de deux relations sur les mêmes ensembles est l'intersection des listes d'associations. id est :* $aR \cap Sb$ *si et seulement si aRb et aSb*

Écrire dans relation.py une fonction *intersection(R,S,et)*
Cette fonction est très similaire à la précédente, il s'agit en fait de la même fonction ! qu'en pensez vous ?

Inclusion *Une relation R sur A,B est inclus dans S sur A,B si la liste d'association de R est inclus dans celle de S. Id Est* $\forall a \in A, b \in B, aRb \Rightarrow aSb$

Écrire dans relation.py une fonction *inclus(R,S)* qui renvoie 1 si la relation R est inclue dans S et 0 sinon.
On rappelle qu'une relation ou un ensemble A est inclus dans un autre B si tous les éléments de A sont inclus dans ceux de B, les taille de R et S peuvent être obtenues via *len*. Faire attention au débordement, si $i > len(R)$, cela signifie que $R[i][*] = 0$, mais causera une erreur dans python.

Tester cette fonction avec S^1 et S^2, S^1 et $S^1 \cup S^2$, *Vide* et *Pleine* , et entre *moitié* et *diviseur*.
(vous devriez obtenir 0, 1, 1, 0, ce dernier car 1 est la moitié de 2 mais 1 est toujours un diviseur trivial)