03_Relations

November 8, 2021

1 Relations et fonctions

Marc Lorenzi

5 novembre 2021

[1]:
```
import matplotlib.pyplot as plt
```

[2]:
```
from hf01 import *
```

1.1 1. Relations

1.1.1 1.1 Introduction

Tout le monde sait ce qu'est une relation. Mais ce que nous allons faire c'est définir *à l'intérieur de l'ensemble* V la notion de relation. Pour pouvoir distinguer nos nouvelles relations des relations usuelles, nous les appellerons des V-relations.

Définition. Une *V-relation* est un ensemble $R \in V$ dont les éléments sont des couples. Pour tous $x, y \in V$, si $(x, y) \in R$ on note $x\, R\, y$.

- Le *domaine* de la V-relation R est $dom(R) = \{x \in V : \exists y \in V, x\, R\, y\}$.
- Le *codomaine* de R est $codom(R) = \{y \in V : \exists x \in V, x\, R\, y\}$.
- Le *champ* de R est $fld(R) = dom(R) \cup codom(R)$.

Si $A \in V$, on dit que R est une relation *sur* A lorsque $fld(R) \subseteq A$.

Proposition. Soit R une V-relation. Alors $dom(R)$, $codom(R)$ et $fld(R)$ appartiennent à V.

Démonstration. $R \in V$, donc R est un ensemble fini. De là, $dom(R)$ et $codom(R)$ sont des parties finies de V, donc des éléments de V. La réunion de deux éléments de V étant encore dans V, on en déduit $fld(R) \in V$. □

La fonction `domaine` prend en paramètre une V-relation R et renvoie $dom(R)$.

[3]:
```
def domaine(R):
    D = []
    for C in R:
        x, y = composantes(C)
        D = reunion(D, [x])
```

```
    return D
```

La fonction `codomaine` prend en paramètre une relation R et renvoie $codom(R)$.

[4]:
```
def codomaine(R):
    D = []
    for C in R:
        x, y = composantes(C)
        D = reunion(D, [y])
    return D
```

La fonction `champ` prend en paramètre une relation R et renvoie $fld(R)$.

[5]:
```
def champ(R):
    return reunion(domaine(R), codomaine(R))
```

1.1.2 1.2 Quelques exemples

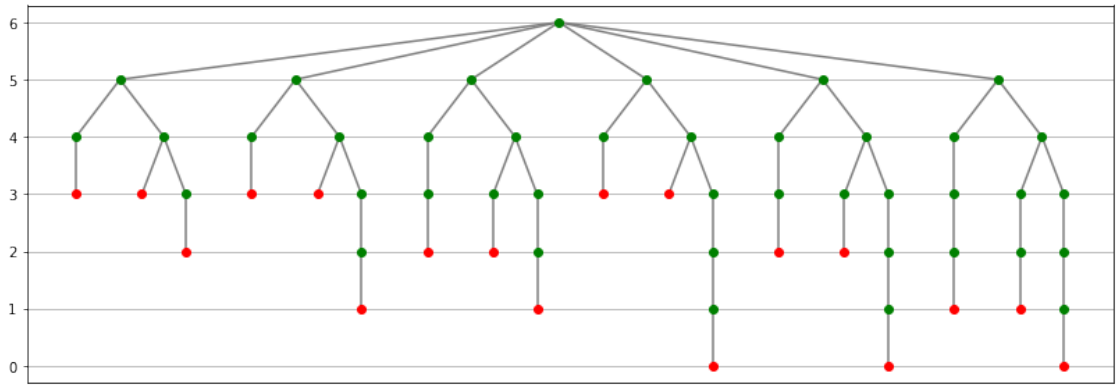
Soit $A \in V$. L'ordre $<$ sur V induit une V-relation sur A.

[6]:
```
def R_inferieur(A):
    R = []
    for x in A:
        for y in A:
            if inferieur_strict(x, y):
                R = reunion(R, [couple(x, y)])
    return R
```

Voici par exemple la relation $<$ sur τ_4.

[7]:
```
R = R_inferieur(tau(4))
print('R        =', tostring(R))
print('dom(R)   = ', tostring(domaine(R)))
print('codom(R) = ', tostring(codomaine(R)))
print('fld(R)   = ', tostring(champ(R)))
tracer_arbre(R)
```

```
R        = {<0,1>,<0, 2>,<1, 2>,<0, 3>,<1, 3>,< 2, 3>}
dom(R)   =  3
codom(R) =  {1, 2, 3}
fld(R)   =  4
```

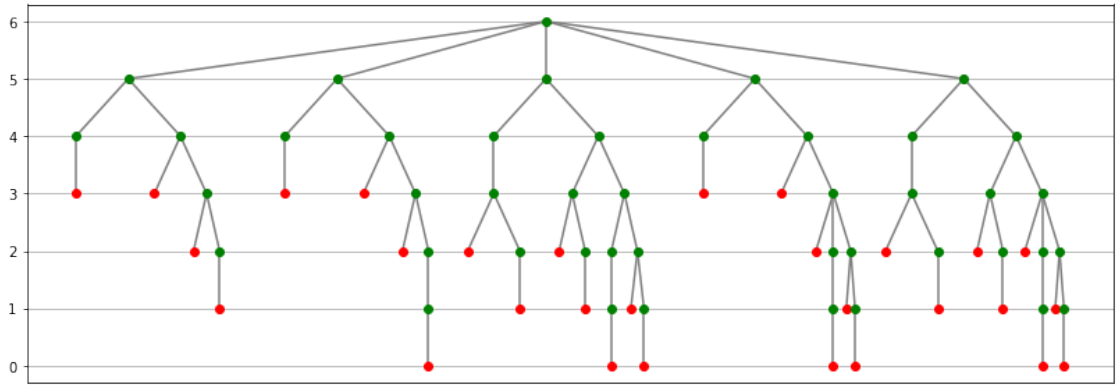


Autre exemple, la relation $\in$ sur V induit une V-relation sur tout ensemble $A \in V$.

[8]:
```
def R_appartient(A):
    R = []
    for x in A:
        for y in A:
            if x in y:
                R = reunion(R, [couple(x, y)])
    return R
```

[9]:
```
R = R_appartient(psi(12345))
print('R        =', tostring(R))
print('dom(R)   =', tostring(domaine(R)))
print('codom(R) =', tostring(codomaine(R)))
print('fld(R)   =', tostring(champ(R)))
tracer_arbre(R)
```

```
R        = {<0,2>,<0,{0, 2}>,<2,<1,0>>,<0,{0, 2,2}>,<2,{0, 2,2}>}
dom(R)   = {0,2}
codom(R) = {2,{0, 2},<1,0>,{0, 2,2}}
fld(R)   = {0,2,{0, 2},<1,0>,{0, 2,2}}
```

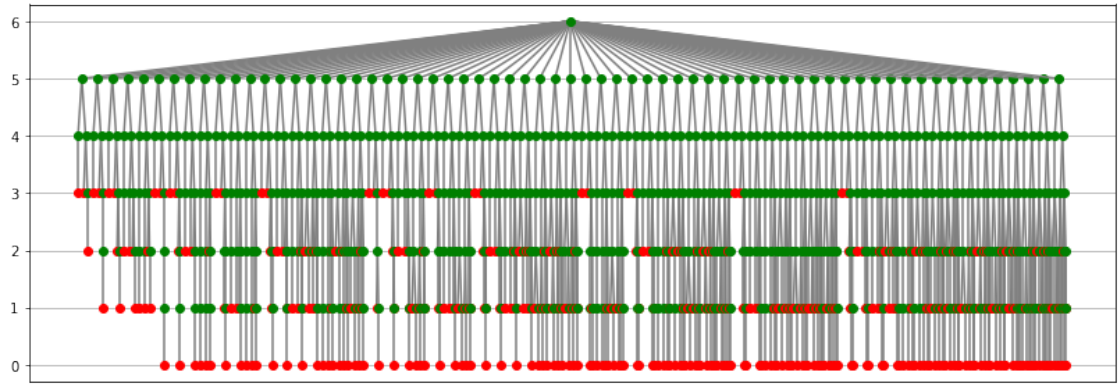


Dernier exemple, la relation $\subset$ sur V induit une V-relation sur tout ensemble $A \in V$.

[10]:
```
def R_inclusion(A):
    R = []
    for x in A:
        for y in A:
            if inclus(x, y) and not x == y:
                R = reunion(R, [couple(x, y)])
    return R
```

[11]:
```
R = R_inclusion(V(4))
#print('R         =', tostring(R))
print('dom(R)   =', tostring(domaine(R)))
print('codom(R) =', tostring(codomaine(R)))
print('fld(R)   =', tostring(champ(R)))
tracer_arbre(R)
```

```
dom(R)   =
{0,1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,2}}
codom(R) = {1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,2}
,{0,1, 2,2}}
fld(R)   = {0,1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,
2},{0,1, 2,2}}
```



1.1.3 1.3 Composition

Soient R et S deux V-relations. La composée de R et S est la V-relation

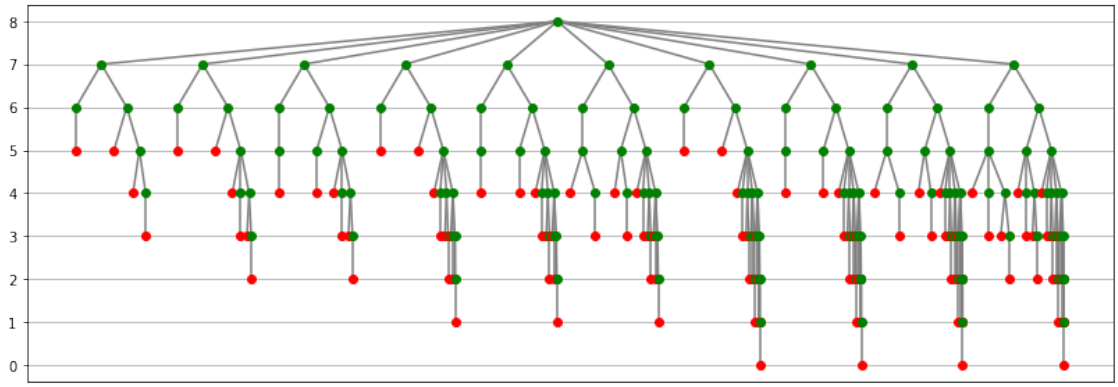
$$S \circ R = \{(x, z) : \exists y \in V, (x, y) \in R \wedge (y, z) \in S\}$$

On a clairement $dom(S \circ R) \subseteq dom(R)$ et $codom(S \circ R) \subseteq codom(S)$.

[12]:
```
def composee(R, S):
    T = []
    for C in R:
        x, y = composantes(C)
        for D in S:
            y1, z = composantes(D)
            if y == y1:
                T = reunion(T, [couple(x, z)])
    return T
```

[13]:
```
R = R_inferieur(neumann(6))
T = composee(R, R)
print(tostring(R))
print(tostring(T))
tracer_arbre(T)
```

```
{<0,1>,<0,2>,<1,2>,<0,3>,<1,3>,<2,3>,<0,4>,<1,4>,<2,4>,<3,4>,<0,5>,<1,5>,<2,5>,<
3,5>,<4,5>}
{<0,2>,<0,3>,<1,3>,<0,4>,<1,4>,<2,4>,<0,5>,<1,5>,<2,5>,<3,5>}
```



1.1.4 1.4 Réciproque d'une relation

Définition. Soit R une V-relation. La *réciproque* de R est la V-relation

$$R^{-1} = \{(y, x) : (x, y) \in R\}$$

[14]:
```
def reciproque(R):
    S = []
    for C in R:
        x, y = composantes(C)
```

```
        D = couple(y, x)
        S = reunion(S, [D])
    return S
```

[15]:
```
R = R_inclusion(V(4))
print(tostring(R))
print()
print(tostring(reciproque(R)))
```

```
{<0,1>,<0, 2>,<0,2>,<1,2>,< 2,2>,<0, 3>,<0,{0, 2}>,<1,{0, 2}>,< 3,{0, 2}>,<0,{1,
 2}>,< 2,{1, 2}>,< 3,{1, 2}>,<0, 3>,<1, 3>,< 2, 3>,<2, 3>,< 3, 3>,<{0, 2}, 3>,<{
1, 2}, 3>,<0,{2}>,<0,{0,2}>,<1,{0,2}>,<{2},{0,2}>,<0,<0,1>>,< 2,<0,1>>,<{2},<0,1
>>,<0,3>,<1,3>,< 2,3>,<2,3>,<{2},3>,<{0,2},3>,<<0,1>,3>,<0,<1,0>>,< 3,<1,0>>,<{2
},<1,0>>,<0,{0, 2,2}>,<1,{0, 2,2}>,< 3,{0, 2,2}>,<{0, 2},{0, 2,2}>,<{2},{0, 2,2}
>,<{0,2},{0, 2,2}>,<<1,0>,{0, 2,2}>,<0,{1, 2,2}>,< 2,{1, 2,2}>,< 3,{1, 2,2}>,<{1
, 2},{1, 2,2}>,<{2},{1, 2,2}>,<<0,1>,{1, 2,2}>,<<1,0>,{1, 2,2}>,<0,{0,1, 2,2}>,<
1,{0,1, 2,2}>,< 2,{0,1, 2,2}>,<2,{0,1, 2,2}>,< 3,{0,1, 2,2}>,<{0, 2},{0,1, 2,2}>
,<{1, 2},{0,1, 2,2}>,< 3,{0,1, 2,2}>,<{2},{0,1, 2,2}>,<{0,2},{0,1, 2,2}>,<<0,1>,
{0,1, 2,2}>,<3,{0,1, 2,2}>,<<1,0>,{0,1, 2,2}>,<{0, 2,2},{0,1, 2,2}>,<{1, 2,2},{0
,1, 2,2}>}

{<1,0>,< 2,0>,<2,0>,<2,1>,<2, 2>,< 3,0>,<{0, 2},0>,<{0, 2},1>,<{0, 2}, 3>,<{1, 2
},0>,<{1, 2}, 2>,<{1, 2}, 3>,< 3,0>,< 3,1>,< 3, 2>,< 3,2>,< 3, 3>,< 3,{0, 2}>,<
3,{1, 2}>,<{2},0>,<{0,2},0>,<{0,2},1>,<{0,2},{2}>,<<0,1>,0>,<<0,1>, 2>,<<0,1>,{2
}>,<3,0>,<3,1>,<3, 2>,<3,2>,<3,{2}>,<3,{0,2}>,<3,<0,1>>,<<1,0>,0>,<<1,0>, 3>,<<1
,0>,{2}>,<{0, 2,2},0>,<{0, 2,2},1>,<{0, 2,2}, 3>,<{0, 2,2},{0, 2}>,<{0, 2,2},{2}
>,<{0, 2,2},{0,2}>,<{0, 2,2},<1,0>>,<{1, 2,2},0>,<{1, 2,2}, 2>,<{1, 2,2}, 3>,<{1
, 2,2},{1, 2}>,<{1, 2,2},{2}>,<{1, 2,2},<0,1>>,<{1, 2,2},<1,0>>,<{0,1, 2,2},0>,<
{0,1, 2,2},1>,<{0,1, 2,2}, 2>,<{0,1, 2,2},2>,<{0,1, 2,2}, 3>,<{0,1, 2,2},{0, 2}>
,<{0,1, 2,2},{1, 2}>,<{0,1, 2,2}, 3>,<{0,1, 2,2},{2}>,<{0,1, 2,2},{0,2}>,<{0,1,
2,2},<0,1>>,<{0,1, 2,2},3>,<{0,1, 2,2},<1,0>>,<{0,1, 2,2},{0, 2,2}>,<{0,1, 2,2},
{1, 2,2}>}
```

Remarquons que si R est une V-relation, alors

- $dom(R^{-1}) = codom(R)$.
- $codom(R^{-1}) = dom(R)$.
- $(R^{-1})^{-1} = R$.

Proposition. Soit R une V-relation de domaine A et de codomaine B. On a alors

- $R \circ R^{-1} \supseteq id_B$.
- $R^{-1} \circ R \supseteq id_A$.

où $id_A = \{(x, x) : x \in A\}$.

Démonstration.

- Soit $x \in B$. Il existe $y \in A$ tel que $y\, R\, x$. On a donc $x\, R^{-1}\, y$ d'où $(x, x) \in R \circ R^{-1}$.
- Même démonstration en posant $S = R^{-1}$. □

Remarquons que l'on a seulement des inclusions, et pas des égalités. Par exemple :

[16]:
```
R = R_inferieur(neumann(5))
print(tostring(domaine(R)), tostring(codomaine(R)))
print(tostring(composee(reciproque(R), R)))
print(tostring(composee(R, reciproque(R))))
```

```
4 {1,2,3,4}
{ 3,<2,2>,<1,2>,<2,1>,<3,3>,<1,3>,<3,1>,<2,3>,<3,2>,<4,4>,<1,4>,<4,1>,<2,4>,<4,2
>,<3,4>,<4,3>}
{ 2, 3,<0,1>,<1,0>,<2,2>,<0,2>,<2,0>,<1,2>,<2,1>,<3,3>,<0,3>,<3,0>,<1,3>,<3,1>,<
2,3>,<3,2>}
```

1.1.5 1.5 Image directe, image réciproque

Définition. Soit R une V-relation. Soit $A \in V$.

- *L'image directe* de A par R est $R[A] = \{y \in V : \exists x \in A, x\, R\, y\}$.
- *L'image réciproque* de A par R est $R^{-1}[A]$, l'image directe de A par R^{-1}.

Facilement, $R[A]$ et $R^{-1}[A]$ appartiennent à V.

[17]:
```
def image_directe(A, R):
    D = []
    for C in R:
        x, y = composantes(C)
        if appartient(x, A):
            D = reunion(D, [y])
    return D
```

[18]:
```
R = R_inferieur(V(4))
print(tostring(image_directe(V(3), R)))
```

```
{1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,2},{0,1, 2,2}
}
```

[19]:
```
def image_reciproque(A, R):
    D = []
    for C in R:
        x, y = composantes(C)
        if appartient(y, A):
            D = reunion(D, [x])
    return D
```

[20]:
```
R = R_inferieur(V(4))
print(tostring(image_reciproque(V(3), R)))
```

```
3
```

1.2 2. Fonctions

1.2.1 2.1 Introduction

Une V-fonction est une V-relation vérifiant une propriété supplémentaire.

Définition. Une *V-fonction* est une V-relation f vérifiant

$$\forall x, y, z \in V, (x, y) \in f \wedge (x, z) \in f \implies y = z$$

Pour tout $x \in dom(f)$, l'unique $y \in V$ tel que $(x, y) \in f$ est appelé *l'image* de x par f, et est noté $f(x)$.

La notation $f : A \longrightarrow B$ signifie que

- f est une fonction.
- $dom(f) = A$.
- $codom(f) \subseteq B$.

La fonction `image` prend en paramètres un élément x de V et une V-fonction f. Si $x \in dom(f)$, elle renvoie $f(x)$. Sinon, elle lève une exception.

[21]:
```
def image(x, f):
    for C in f:
        A, B = composantes(C)
        if x == A: return B
    raise Exception('Pas dans le domaine')
```

1.2.2 2.2 Un exemple

Prenons juste un exemple. Soit $A \in V$. La fonction « successeur de domaine A » est la V-fonction $f : A \longrightarrow V$ définie par $f(x) = x^+$.

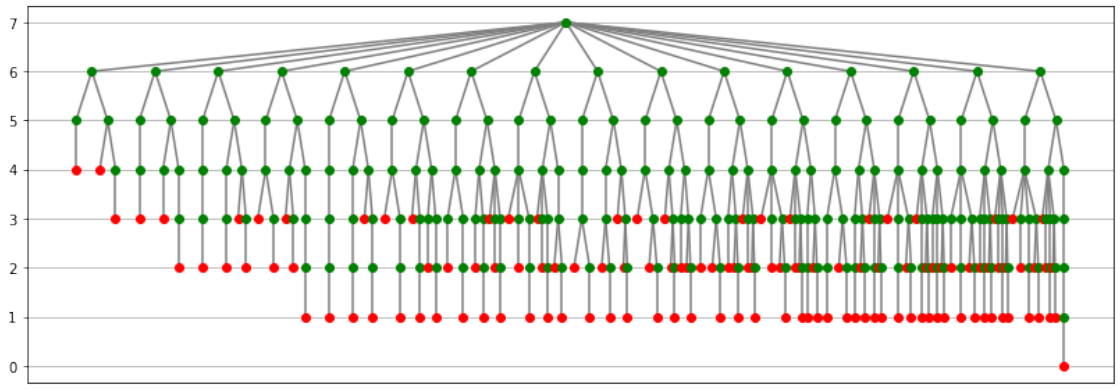
[22]:
```
def f_succ(A):
    f = []
    for x in A:
        f = reunion(f, [couple(x, succ(x))])
    return f
```

Voici la fonction successeur de domaine V_4.

[23]:
```
SV4 = f_succ(V(4))
print(tostring(SV4))
```

```
{<0,1>,<1, 2>,< 2,2>,<2, 3>,< 3,{0, 2}>,<{0, 2},{1, 2}>,<{1, 2}, 3>,< 3,{2}>,<{2
},{0,2}>,<{0,2},<0,1>>,<<0,1>,3>,<3,<1,0>>,<<1,0>,{0, 2,2}>,<{0, 2,2},{1, 2,2}>,
<{1, 2,2},{0,1, 2,2}>,<{0,1, 2,2}, 4>}
```

[24]:
```
tracer_arbre(SV4)
```



1.2.3 2.3 Composition

Proposition. La composée de deux V-fonctions est une V-fonction.

Démonstration. Laissée en exercice. Si f et g sont deux V-fonctions, on vérifie aisément que

$$dom(g \circ f) = \{x \in dom(f) : f(x) \in dom(g)\} = f^{-1}[dom(g)]$$

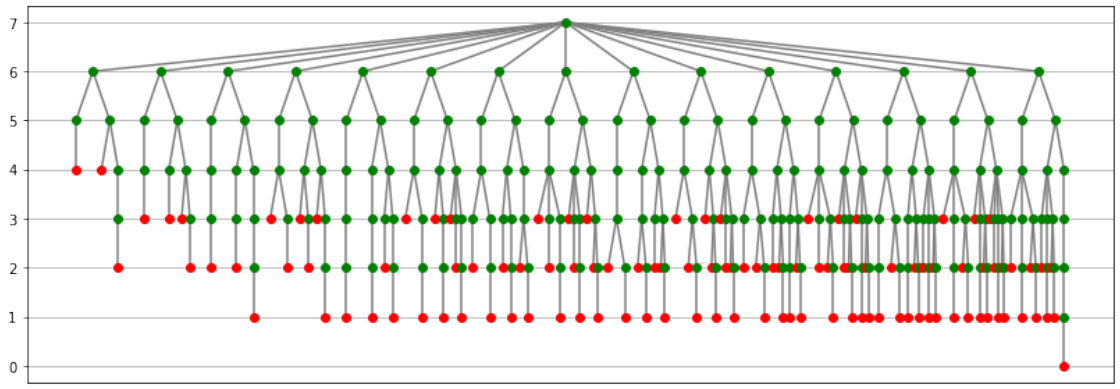
et

$$codom(g \circ f) = g[dom(g) \cap codom(f)]$$

On a alors pour tout $x \in dom(g \circ f)$, $(g \circ f)(x) = g(f(x))$.

[25]:
```
h = composee(SV4, SV4)
print(tostring(h))
tracer_arbre(h)
```

```
{<0, 2>,<1,2>,< 2, 3>,<2,{0, 2}>,< 3,{1, 2}>,<{0, 2}, 3>,<{1, 2},{2}>,< 3,{0,2}>
,<{2},<0,1>>,<{0,2},3>,<<0,1>,<1,0>>,<3,{0, 2,2}>,<<1,0>,{1, 2,2}>,<{0, 2,2},{0,
1, 2,2}>,<{1, 2,2}, 4>}
```



[26]:
```
print(tostring(domaine(h)))
print(tostring(codomaine(h)))
```

```
{0,1, 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,2}}
{ 2,2, 3,{0, 2},{1, 2}, 3,{2},{0,2},<0,1>,3,<1,0>,{0, 2,2},{1, 2,2},{0,1, 2,2},
4}
```

1.2.4 2.4 Injections

La réciproque d'une V-fonction n'est pas en général une V-fonction, mais seulement une V-relation.

Proposition. Soit f une V-fonction. f^{-1} est une V-fonction si et seulement si

$$\forall x, x' \in dom(f), f(x) = f(x') \implies x = x'$$

Démonstration. f^{-1} est une V-fonction si et seulement si

$$\forall x, x', y \in V, (y, x) \in f^{-1} \wedge (y, x') \in f^{-1} \implies x = x'$$

ou encore

$$\forall x, x', y \in V, (x, y) \in f \wedge (x', y) \in f \implies x = x'$$

Il est facile de voir que ceci équivaut à la propriété voulue.

Une fonction vérifiant cette propriété est appelée une *injection*.

1.2.5 2.5 Surjections, bijections

Soit $f : A \longrightarrow B$ une V-fonction. La fonction f est une *surjection* de A *sur* B si $codom(f) = B$. Remarquons que, contrairement à l'injectivité, la notion de surjectivité est une sorte de pléonasme puisque **toute** fonction est une surjection de son domaine sur son codomaine.

La V-fonction $f : A \longrightarrow B$ est une *bijection* si elle est injective et surjective.

Soient $A, B \in V$. A et B sont *équipotents* s'il existe une bijection de A sur B. On note $A \simeq B$. Les éléments de V étant des ensembles finis, on a $A \simeq B$ si et seulement si $|A| = |B|$.

Nous ne creuserons pas davantage. Nous avons introduit dans l'univers V toutes les notions usuelles, maintenant on peut faire à l'intérieur de V des mathématiques « comme d'habitude ». Enfin, presque, parce que l'univers V ne contient pas d'objets « infinis » …

Notons tout de même le résultat suivant.

Proposition. Soit $A \in V$. A est équipotent à un unique entier de von Neumann $\overline{n}$.

Il existe aussi un unique $n \in \mathbb{N}$ tel que $A \simeq \tau_n$. Mais les théoriciens des ensembles préfèrent les entiers de von Neumann aux ensembles τ_n. Pourquoi ? Parce que lorsqu'on s'intéresse à ce qui se passe *au-delà* de V, pour des ensembles infinis, les ensembles τ_n ne se généralisent pas, alors que les entiers de von Neumann continuent à exister. Rappelons que pour tout $n \in \mathbb{N}$,

$$\overline{n+1} = \overline{n} \cup \{\overline{n}\}$$

On peut poser $\omega = \{\overline{n} : n \in \mathbb{N}\}$, puis considérer $S(\omega) = \omega \cup \{\omega\}$ et appeler cet ensemble le successeur de ω. Rien n'empêche de continuer, et cela débouche sur ce que l'on appelle la théorie des ordinaux … mais ceci est une autre histoire.

1.3 3. Bilan

1.3.1 3.1 V et la théorie des ensembles

Dans ce notebook et les précédents, nous avons vu que *l'univers* V vérifie des propriétés intéressantes. En fait, V satisfait à presque toutes les propriétés qui constituent les *axiomes* de la théorie des ensembles de Zermelo-Fraenkel. Nous allons passer en revue ces axiomes. Pour ne pas être trop formalistes, nous formulerons ces axiomes de façon naïve.

Commençons par l'axiome d'extensionalité.

Extensionalité. Soient $A, B \in V$. Si A et B ont les mêmes éléments, alors ils sont égaux.

Cet axiome est vérifié pour **tous** les ensembles, et donc aussi pour les ensembles de l'univers V. Passons à des axiomes un peu moins évidents.

1.3.2 3.2 Paire, réunion, parties

Certains des axiomes de la théorie des ensembles affirment que si certains objets sont des ensembles, alors d'autres en sont aussi. En voici trois :

Paire. Si $A, B \in V$, alors $\{A, B\} \in V$.

Réunion. Si $A \in V$, alors $\bigcup_{x \in A} x \in V$.

Parties. Si $A \in V$, alors $\mathcal{P}(A) \in V$.

Ces propriétés nous disent que l'univers V satisfait les axiomes de la *paire*, de la *réunion* et des *parties*.

Remarquons que les objets dont l'existence est assurée par ces trois propriétés peuvent être construits de façon *effective* par les fonctions `PAIRE`, `UNION` et `PARTIES` ci-dessous.

[27]:
```
def PAIRE(A, B):
    if inferieur_strict(A, B): return [A, B]
    elif inferieur_strict(B, A): return [B, A]
    else: return [A]
```

[28]:
```
print(tostring(PAIRE(tau(6), neumann(4))))
```

```
{4, 6}
```

[29]:
```
def UNION(A):
    U = []
    for x in A:
        U = reunion(U, x)
    return U
```

[30]:
```
A = psi(123456)
print(tostring(A))
print(tostring(UNION(A)))
```

```
{{1, 2},{0,2},{0, 2,2},{1, 2,2},{0,1, 2,2}, 4}
{0,1, 2,2, 3}
```

[31]:
```
def PARTIES(A): return parties(A)
```

[32]:
```
A = psi(1234)
print(tostring(A))
print(tostring(PARTIES(A)))
```

```
{1, 3,{1, 2}, 3,<0,1>}
{0, 2, 4,{1, 3},{{1, 2}},{1,{1, 2}},< 2,1>,{1, 3,{1, 2}},{ 3},{1, 3},{ 3, 3},{1,
 3, 3},{{1, 2}, 3},{1,{1, 2}, 3},{ 3,{1, 2}, 3},{1, 3,{1, 2}, 3},{<0,1>},{1,<0,1
>},{ 3,<0,1>},{1, 3,<0,1>},{{1, 2},<0,1>},{1,{1, 2},<0,1>},{ 3,{1, 2},<0,1>},{1,
 3,{1, 2},<0,1>},{ 3,<0,1>},{1, 3,<0,1>},{ 3, 3,<0,1>},{1, 3, 3,<0,1>},{{1, 2},
3,<0,1>},{1,{1, 2}, 3,<0,1>},{ 3,{1, 2}, 3,<0,1>},{1, 3,{1, 2}, 3,<0,1>}}
```

1.3.3 3.2 Formules logiques

Nous allons dans ce paragraphe être très succincts, nous ne donnerons pas tous les détails. On suppose donné un ensemble de *variables* x, y, z, etc. Les variables sont censées représenter des ensembles héréditairement finis.

Définissons récursivement la notion de *formule logique*.

- Si x et y sont deux variables, $x \in y$ et $x = y$ sont des formules, dites *atomiques*.
- Si φ et ψ sont deux formules, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \implies \psi$, $\varphi \iff \psi$ et $\neg\varphi$ sont aussi des formules.

Nous supposons que le lecteur est familiarisé avec la notion de variable libre et de variable liée. Une formule est dite *close* si elle ne contient pas de variable libre. Par exemple, tous les axiomes de la théorie des ensembles sont des formules closes.

Si φ est une formule, nous noterons $\varphi(x)$ pour signaler que x figure éventuellement parmi les variables libres de φ. De même avec $\varphi(x, y)$, etc.

Nous allons discuter ici d'une notion que nous ne formaliserons pas de façon totalement rigoureuse. Nous mettons deux paramètres aux formules, mais il pourrait y en avoir 1 ou 3 ou 4, etc.

Définition. Soit $\varphi(x, y)$ une formule. Le formule φ est *décidable* s'il existe une fonction Python f_φ telle que pour tous $x, y \in V$ l'appel $f_\varphi(x, y)$ renvoie `True` si $\varphi(x, y)$ est satisfaite et `False` sinon. Dit autrement, il existe un *algorithme* qui permet de savoir si φ est satisfaite par les ensembles x et y de V.

Savoir si une formule φ donnée est décidable n'est pas une question facile. Juste pour estimer la difficulté du problème, tous les théorèmes des mathématiques sont des formules. Si toutes les formules étaient décidables, alors tous les théorèmes le seraient aussi. Nous serions donc en mesure de prouver tous les théorèmes par des moyens purement mécaniques. Si l'univers dans lequel nous travaillons est l'univers classique, un théorème dû à Kurt Gödel affirme qu'il existe des formules qui ne sont pas décidables. Quant à savoir si l'univers des « mathématiques dans V » contient des formules indécidables, je laisse la question ouverte. J'ai trop peur de dire des bêtises .

1.3.4 3.3 Compréhension

Le *schéma d'axiomes de compréhension* est la propriété suivante.

Compréhension. Soit $\varphi(x)$ une formule dépendant de la variable x. Pour tout $A \in V$, il existe $B \in V$ tel que

$$\forall x \in V,\ x \in B \iff x \in A \wedge \varphi(x)$$

Démonstration. Soit $B = \{x \in A : \varphi(x)\}$. On a $B \subseteq A$ et $A \in V$, donc $B \in V$. Et B satisfait clairement la propriété voulue. □

Si la formule φ est décidable, l'ensemble B peut être construit de façon *effective*. Soit `f_phi` une fonction qui renvoie `True` si $\varphi(x)$ est satisfaite par $x \in V$, et `False` sinon.

[33]:
```
def COMPREHENSION(A, f_phi):
    B = []
    for x in A:
        if f_phi(x):
            B = reunion(B, [x])
    return B
```

Par exemple, soit $\mathcal{P} = \{x \in V : \varphi(x) \equiv 0 \bmod 5\}$ (la fonction φ est ici notre bijection $V \longrightarrow \mathbb{N}$). Soit φ la formule logique $\varphi(x) = x \in \mathcal{P}$. Il n'est absolument pas évident que φ soit une formule

logique, mais passons. Soit $A = \psi(123456789^4)$. Que vaut $\{x \in A : \psi(x)\}$?

[34]:
```
def f_phi(x): return phi(x) % 5 == 0
```

[35]:
```
A = psi(123456789 ** 4)
B = COMPREHENSION(A, f_phi)
for x in B:
    print(phi(x), tostring(x))
```

```
0 0
5 {0, 2}
25 {0,2, 3}
30 {1, 2,2, 3}
50 {1, 3,{0, 2}}
65 {0,{1, 2}}
70 {1, 2,{1, 2}}
90 {1,2, 3,{1, 2}}
100 { 2,{0, 2},{1, 2}}
105 {0,2,{0, 2},{1, 2}}
```

1.3.5 3.4 Remplacement

Une formule $\varphi(x, y)$ dépendant des variables x et y est dite *fonctionnelle* en x si

$$\forall x\ \forall y\ \forall z\ \varphi(x, y) \wedge \varphi(x, z) \implies y = z$$

Remplacement. Pour toute formule fonctionnelle φ, pour tout $A \in V$, il existe un ensemble $B \in V$ tel que

$$\forall y \in V\ y \in B \iff \exists x \in A\ \varphi(x, y)$$

Démonstration. Il suffit de considérer $B = \{y \in V : \exists x \in A\ \varphi(x, y)\}$. L'ensemble B est inclus dans V, et comme φ est fonctionnelle, B est un ensemble fini, de cardinal au plus égal à $|A|$. Ainsi, $B \in V$. □

Ici aussi, si φ est décidable, on peut écrire une fonction qui calcule l'ensemble B. Cette fonction est totalement inefficace, mais elle prouve la *calculabilité* de B. En vertu du schéma d'axiomes de remplacement, on sort obligatoirement de la boucle `while`.

[36]:
```
def REMPLACEMENT(A, f_phi):
    B = []
    while True:
        if solution(A, B, f_phi): return B
        B = succ(B)
```

La fonction `solution` renvoie `True` si B convient, et `False` sinon.

- La boucle #1 regarde si $B \subseteq \{y \in V : \exists x \in A, \varphi(x, y)\}$, c'est à dire $\forall y \in B, \exists x \in A, \varphi(x, y)$.

- La boucle #2 regarde si $\forall x \in A, \exists y \in B, \varphi(x, y)$. Comme φ est fonctionnelle, cela prouve que $B \supseteq \{y \in V : \exists x \in A, \varphi(x, y)\}$.

[37]:
```
def solution(A, B, f_phi):
    for y in B: #1
        ok = False
        for x in A:
            if f_phi(x, y):
                ok = True
                break
        if not ok: return False
    for x in A: #2
        ok = False
        for y in B:
            if f_phi(x, y):
                ok = True
                break
        if not ok: return False
    return True
```

Tester la fonction `REMPLACEMENT` sur un exemple relève du défi, ou alors prenons un exemple trèèès simple.

Prenons $\varphi(x, y) =$ « $y = x \cup \overline{2}$ ».

[38]:
```
def f_phi(x, y):
    return y == reunion(x, tau(2))
```

Appliquons à τ_5.

[39]:
```
print(tostring(REMPLACEMENT(tau(5), f_phi)))
```

```
{2, 3,{0,1, 3}}
```

Évidemment, cet exemple est trooop simple. On peut retrouver le résultat en 1 ligne :

[40]:
```
for x in tau(5):
    print(tostring(reunion(x, tau(2))), end=' - ')
```

```
2 - 2 - 2 -  3 - {0,1, 3} -
```

Cela dit, cela montre que notre fonciton `REMPLACEMENT` n'est pas si absurde que cela.

1.3.6 3.5 Infini

L'univers V ne satisfait pas l'axiome de l'infini.

Définition. Soit $A \in V$. L'ensemble A est *inductif* lorsque

- $\emptyset \in A$.

- $\forall x \in V,\ x \in A \implies x \cup \{x\} \in A$.

¬ **Infini.** Soit $A \in V$. Alors A n'est pas inductif.

Démonstration. Supposons que A est inductif. Alors, facilement, A contient tous les entiers de von Neumann. Oui, mais … Soit n le rang de A. Alors $\overline{n} \notin A$ puisque $\operatorname{rg} \overline{n} = n \not< \operatorname{rg} A$. Contradiction. □

Il reste deux axiomes à examiner, que certains pourraient qualifier de « mystérieux » : *l'axiome de fondation* et *l'axiome du choix*. Nous allons ici aussi montrer que dans l'univers V ces axiomes sont vérifiés de façon *constructive*. Espérons qu'après cela ces axiomes seront un peu moins mystérieux.

1.3.7 3.6 L'axiome de fondation

Fondation. Soit $A \in V$ non vide. Il existe $x \in A$ tel que pour tout $y \in V$, $y \in x \implies y \notin A$.

l'axiome de fondation nous dit que tout ensemble non vide héréditairement fini possède un *élément minimal* pour la relation d'appartenance.

Démonstration. Il suffit de prendre pour x un élément de A de rang minimal. □

Remarquons que le plus petit élément de A (pour la relation $\leq$) convient. On a donc immédiatement une fonction `fondation` qui prend en paramètre un ensemble $A \in V$ non vide et renvoie un élément x de A minimal pour la relation d'appartenance.

[41]:
```
def fondation(A): return A[0]
```

[42]:
```
print(tostring(fondation(psi(109876))))
```

```
2
```

Remarquons qu'un ensemble $A \in V$ peut posséder plusieurs éléments minimaux pour la relation d'appartenance. La fonction `minimaux` les renvoie tous.

[43]:
```
def minimaux(A):
    if len(A) == 1: return [A[0]]
    else:
        M = minimaux(A[1:])
        x = A[0]
        return [x] + [y for y in M if not appartient(x, y)]
```

[44]:
```
A = psi(123456789)
print('A =', tostring(A))
print('Éléments minimaux de A :', end=' ')
for x in minimaux(A):
    print(tostring(x), end=', ')
```

```
A = {0, 2, 3,{2},<0,1>,3,{1, 2,2},{0,1, 2,2}, 4,{0, 3},{0,1, 3},{ 2, 3},{1, 2, 3
},{2, 3},{0,2, 3},{1,2, 3}}
Éléments minimaux de A : 0,  2, {2}, <0,1>,
```

1.3.8 3.7 L'axiome du choix

Choix. Soit $A \in V$ ne contenant pas $\emptyset$. Il existe une V-fonction $f : A \longrightarrow V$ telle que pour tout $x \in A$, $f(x) \in x$.

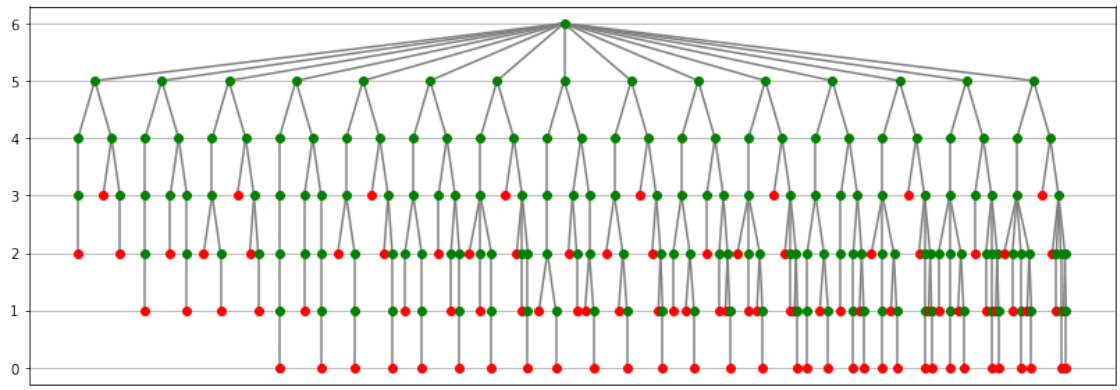
Une telle fonction f est appelée une *fonction de choix* pour l'ensemble A.

[45]:
```
def choix(A):
    f = []
    for x in A:
        f = reunion(f, [couple(x, x[0])])
    return f
```

Voici par exemple une fonction de choix pour $V_4 \setminus \{\emptyset\}$.

[46]:
```
f = choix(difference(V(4), [[]]))
print(tostring(f))
tracer_arbre(f)
```

```
{<1,0>,< 2,1>,<2,0>,< 3, 2>,<{0, 2},0>,<{1, 2},1>,< 3,0>,<{2},2>,<{0,2},0>,<<0,1
>,1>,<3,0>,<<1,0>, 2>,<{0, 2,2},0>,<{1, 2,2},1>,<{0,1, 2,2},0>}
```



En fait, nous pouvons faire encore mieux. L'univers V possède une *fonction de choix global*, c'est à dire une fonction $\Gamma : V \setminus \{\emptyset\} \longrightarrow V$ telle que pour tout $x \in V \setminus \{\emptyset\}$, $\Gamma(x) \in x$. Un exemple de telle fonction est la fonction définie par $\Gamma(x) = \min x$.

Remarquons cependant qu'une telle fonction n'est **pas** une V-fonction : l'ensemble $\Gamma = \{(x, \Gamma(x)) : x \in V \setminus \{\emptyset\}\}$ est une partie de V, mais cette partie est infinie. C'est ce que nous appellerons dans la dernière section de ce notebook une classe propre. La fonction `choix_global` ci-dessous *simule* le comportement de Γ. Il est bien évident que nous ne pouvons pas écrire du code python qui *calcule* l'ensemble infini Γ ...

[47]:
```
def choix_global(x): return x[0]
```

[48]:
```
print(tostring(choix_global(psi(123456))))
```

```
{1, 2}
```

1.3.9 3.8 Conclusion

L'univers V satisfait donc tous les axiomes de la théorie des ensembles, *sauf* l'axiome de l'infini. Encore, mieux, nous avons vu qu'il est possible de construire de façon *effective* (bref, de programmer !) les objets dont l'existence est assurée par les axiomes, du moins dans certains cas.

Nous allons terminer ce notebook par une discussion portant sur les parties de V qui ne sont pas des éléments de V …

1.4 4. Classes

1.4.1 4.1 Introduction

Dans la théorie des ensembles, les *classes* sont des « collections d'objets ». Plus précisément, toute formule logique $\varphi(x)$ définit la classe

$$C = \{x : \varphi(x)\}$$

Un ensemble x *appartient* à la classe C si et seulement si $\varphi(x)$ est vérifiée. On note évidemment $x \in C$.

Certaines classes « sont » des ensembles, d'autres pas. Une classe qui n'est pas un ensemble est ce que l'on appelle une *classe propre*. Dans l'univers V des ensembles héréditairement finis, il est très facile de trancher. Une classe est une partie de V de la forme $C = \{x \in V : \varphi(x)\}$. Quelles sont les classes propres ? Facile ! En effet, soit $C \subseteq V$. Nous avons vu dans le premier notebook que

$$C \in V \iff C \text{ est fini}$$

Les classes propres sont donc les parties infinies de V. Par exemple, V lui-même est une classe propre.

1.4.2 4.2 Toutes les classes propres sont équipotentes

Remarquons que, V étant dénombrable (c'est à dire en bijection avec les entiers naturels), un résultat classique affirme que toutes les parties infinies de V sont aussi dénombrables. Ainsi, toutes les classes propres sont en bijection les unes avec les autres.

Notons $\omega = \{\overline{n} : n \in \mathbb{N}\}$.

ω est une classe propre, c'est la classe des entiers de von Neumann, que l'on appelle aussi les *ordinaux finis*. Le lecteur pourrait ne pas être convaincu et se demander : mais quelle serait une formule φ qui définit ω ? Eh bien nous avons vu qu'un ensemble héréditairement fini est un entier de von Neumann si et seulement si il est bitransitif. Or, il est facile d'exprimer la bitransitivité par une formule.

Un ensemble $x \in V$ est transitif si et seulement si $trans(x)$, où

$$trans(x) \iff \forall y \in x, \forall z \in y, z \in x$$

Il est bitransitif si et seulement si $bitrans(x)$, où

$$bitrans(x) \iff trans(x) \wedge \forall y \in x, trans(y)$$

Ainsi,

$$\omega = \{x : bitrans(x)\}$$

Toutes les classes propres de V sont ainsi en bijection avec ω.

Proposition. Soit C une classe propre. Il existe $F : \omega \longrightarrow C$ bijective.

Quand nous parlons d'une bijection F, F est un ensemble de couples, et donc $F \subseteq V$, mais attention ! F est un ensemble *infini* de couples, ce n'est donc pas une V-fonction mais ce que nous pourrions appeler une fonction propre. Une telle bijection F est facile à fabriquer. Remarquons tout d'abord le résultat suivant :

Proposition. Soit C une classe non vide. Alors C possède un plus petit élément pour la relation $\leq$.

Démonstration. L'ensemble $\{\varphi(x) : x \in C\}$ est une partie non vide de $\mathbb{N}$, qui possède donc un plus petit élément.

Soit C une classe propre. Posons $C_0 = \min C$ puis, pour tout $n \in \mathbb{N}^*$,

$$C_n = \min(C \setminus \{C_0, \ldots, C_{n-1}\})$$

Sans entrer dans les détails, on définit ainsi une fonction $F : \omega \longrightarrow C$ en posant pour tout $n \in \mathbb{N}$, $F(\overline{n}) = C_n$. On montre alors que F est une bijection.

Reamrquons pour terminer que notre bijection F est un ensemble infini, inclus dans V et donc que F une classe propre, et donc que F est en bijection avec ω, la bijection en question étant aussi une classe propre, etc …

1.4.3 4.3 Il y a beaucoup de classes propres

Remarquons qu'il existe « beaucoup plus » de classes propres que d'ensembles héréditairement finis. En effet, V est dénombrable, alors que $\mathcal{P}(V)$ ne l'est pas. Il y a donc « beaucoup plus » de parties infinies de V que de parties finies de V.

1.4.4 4.4 Et les classes de classes ?

Une classe est une partie de V, ses éléments ne peuvent donc pas être des classes propres mais seulement des éléments de V. Peut-on parler de la classe de toutes les classes ? En tant qu'êtres supérieurs, nous savons que V n'est pas la fin de l'histoire. Rien ne nous empêche de sortir de V et de considérer l'ensemble $\mathcal{P}(V)$. On a

$$\mathcal{P}(V) = V \cup \mathcal{C}$$

où $V = \mathcal{P}_{<\infty}(V)$ est l'ensemble des parties *finies* de V et $\mathcal{C} = \mathcal{P}_{\infty}(V)$ est l'ensemble des parties *infinies* de V, c'est à dire l'ensemble des classes propres. La « classe » de toutes les classes propres serait $\mathcal{C}$ mais remarquons que $\mathcal{C}$ n'est pas une partie de V, ce n'est donc pas une classe. En fait, $\mathcal{C}$ est une partie de $\mathcal{P}(V)$, c'est à dire un élément de $\mathcal{P}(\mathcal{P}(V))$. Une super-classe, peut-être ?

Alors oui, pourquoi pas, parlons de super-classes dont les éléments sont des classes. Et pourquoi pas de super-super-classes dont les éléments sont des super-classes, et ainsi de suite. Mais il y a bien longtemps que nous avons quitté le monde rassurant des ensembles hérédiairement finis …