Lois, structures, matrices en informatique

INFO2 - Semaines 43 & 45

Guillaume CONNAN

IUT de Nantes - Dpt d'informatique

Dernière mise à jour : 24 octobre 2013 à 14:48

Sommaire

Sommaire

Définition 1 (Loi de composition interne (LCI))

Une loi de composition interne définie sur un ensemble E est une fonction totale (ou application) de $E \otimes E$ dans E. On dit alors que E est un **magma**.

Définition 2 (Stabilité)

Un ensemble E est stable par une opération $\star$ si, et seulement si :

$$\Big(\forall \langle x, y\rangle\Big)\Big(\big(\langle x, y\rangle \in E \otimes E\big) \rightarrow \big(x \star y \in E\big)\Big)$$

Définition 3 (Loi de composition externe (LCE))

Une loi de composition externe définie sur un ensemble E et à opérateurs dans un ensemble K est une fonction totale (ou application) de $K \otimes E$ dans E.

Définition 4 (Morphisme)

Soit $\langle E, \star \rangle$ et $\langle F, \dagger \rangle$ deux magmas et φ une fonction totale de E dans F. On dit que φ est un morphisme de E dans F si, et seulement si :

$$(\forall x)(\forall y)(\varphi(x \star y) = \varphi(x) \dagger \varphi(y))$$

Définition 5 (Isomorphisme)

Soit $\langle E, \star \rangle$ et $\langle F, \dagger \rangle$ deux magmas et φ une fonction totale de E dans F. On dit que φ est un morphisme de E dans F si, et seulement si, c'est un morphisme BIJECTIF de E dans F.

Définition 6 (Commutativité)

Une LCI $\star$ sur E est commutative si, et seulement si

$$(\forall x)(\forall y)(x \star y = y \star x)$$

Définition 7 (Associativité)

Une LCI $\star$ sur E est associative si, et seulement si

$$(\forall x)(\forall y)(\forall z)(x \star (y \star z) = (x \star y) \star z)$$

Un magma muni d'une loi associative est appelé... un magma associatif.

Définition 8 (Élément neutre)

Un élément neutre e d'une LCI $\star$ sur E est un élément $e_\star$ qui vérifie :

$$(\forall x)(x \star e_\star = e_\star \star x = x)$$

Un magma associatif admettant un élément neutre est un **monoïde** (ou un magma associatif **unifère**).

Pouvez-vous démontrer que si l'élément neutre existe, il est unique ?

Définition 8 (Élément neutre)

Un élément neutre e d'une LCI $\star$ sur E est un élément $e_\star$ qui vérifie :

$$(\forall x)(x \star e_\star = e_\star \star x = x)$$

Un magma associatif admettant un élément neutre est un **monoïde** (ou un magma associatif **unifère**).

Pouvez-vous démontrer que si l'élément neutre existe, il est unique ?

Définition 8 (Élément neutre)

Un élément neutre e d'une LCI $\star$ sur E est un élément $e_\star$ qui vérifie :

$$(\forall x)(x \star e_\star = e_\star \star x = x)$$

Un magma associatif admettant un élément neutre est un **monoïde** (ou un magma associatif **unifère**).

Pouvez-vous démontrer que si l'élément neutre existe, il est unique ?

Définition 9 (Élément symétrisable)

Soit x un élément de E. Il est inversible (ou symétrisable) par $\star$ si, et seulement si,

$$(\exists y)(x \star y = y \star x = e_\star)$$

Pouvez-vous démontrer que si x admet un symétrique, alors ce symétrique est unique ?

Définition 9 (Élément symétrisable)

Soit x un élément de E. Il est inversible (ou symétrisable) par $\star$ si, et seulement si,

$$(\exists y)(x \star y = y \star x = e_\star)$$

Pouvez-vous démontrer que si x admet un symétrique, alors ce symétrique est unique ?

Définition 9 (Élément symétrisable)

Soit x un élément de E. Il est inversible (ou symétrisable) par $\star$ si, et seulement si,

$$(\exists y)(x \star y = y \star x = e_\star)$$

Pouvez-vous démontrer que si x admet un symétrique, alors ce symétrique est unique ?

Définition 10 (Régularité)

Un élément a de E est dit **régulier à gauche** (ou **simplifiable à gauche**) si, et seulement si :

$$(\forall x)(\forall y)((a \star x = a \star y) \rightarrow (x = y))$$

On a une définition similaire de la régularité à droite.
Un élément à la fois régulier à gauche et à droite est dit **régulier**.
Une loi telle que tout élément soit régulier est dite régulière.

Un élément inversible est régulier (vérifiez-le). Un élément régulier est-il inversible ?

Définition 10 (Régularité)

Un élément a de E est dit **régulier à gauche** (ou **simplifiable à gauche**) si, et seulement si :

$$(\forall x)(\forall y)((a \star x = a \star y) \rightarrow (x = y))$$

On a une définition similaire de la régularité à droite.
Un élément à la fois régulier à gauche et à droite est dit **régulier**.
Une loi telle que tout élément soit régulier est dite régulière.

Un élément inversible est régulier (vérifiez-le). Un élément régulier est-il inversible ?

Définition 10 (Régularité)

Un élément a de E est dit **régulier à gauche** (ou **simplifiable à gauche**) si, et seulement si :

$$(\forall x)(\forall y)((a \star x = a \star y) \rightarrow (x = y))$$

On a une définition similaire de la régularité à droite.
Un élément à la fois régulier à gauche et à droite est dit **régulier**.
Une loi telle que tout élément soit régulier est dite régulière.

Un élément inversible est régulier (vérifiez-le). Un élément régulier est-il inversible ?

Définition 11 (Distributivité)

On dit que la loi $\star$ définie sur E est distributive sur $\dagger$ définie sur E si, et seulement si :

$$\Big(\forall x\Big)\Big(\forall y\Big)\Big(\forall z\Big)\Big(\big(x\star(y\dagger z)=(x\star y)\dagger(x\star z)\big)\wedge\big((y\dagger z)\star x=(y\star x)\dagger(z\star x)\big)\Big)$$

Définition 12 (Élément absorbant)

Un élément absorbant d'une LCI $\star$ sur E est un élément $a_\star$ qui vérifie :

$$(\forall x)(x \star a_\star = a_\star \star x = a_\star)$$

Montrez que si un tel élément existe, il est unique.

Définition 12 (Élément absorbant)

Un élément absorbant d'une LCI $\star$ sur E est un élément $a_\star$ qui vérifie :

$$(\forall x)(x \star a_\star = a_\star \star x = a_\star)$$

Montrez que si un tel élément existe, il est unique.

Définition 12 (Élément absorbant)

Un élément absorbant d'une LCI $\star$ sur E est un élément $a_\star$ qui vérifie :

$$(\forall x)(x \star a_\star = a_\star \star x = a_\star)$$

Montrez que si un tel élément existe, il est unique.

Définition 13 (Élément involutif)

Soit $\star$ une loi sur un ensemble E admettant un élément neutre $e_\star$.
Alors x est **involutif** si, et seulement si, $x \star x = e_\star$.

Définition 14 (Élément idempotent)

Soit $\star$ une loi sur un ensemble E.
Alors x est **idempotent** si, et seulement si, $x \star x = x$.

Définition 13 (Élément involutif)

Soit $\star$ une loi sur un ensemble E admettant un élément neutre $e_\star$.
Alors x est **involutif** si, et seulement si, $x \star x = e_\star$.

Définition 14 (Élément idempotent)

Soit $\star$ une loi sur un ensemble E.
Alors x est idempotent si, et seulement si, $x \star x = x$.

Définition 13 (Élément involutif)

Soit $\star$ une loi sur un ensemble E admettant un élément neutre $e_\star$.
Alors x est **involutif** si, et seulement si, $x \star x = e_\star$.

Définition 14 (Élément idempotent)

Soit $\star$ une loi sur un ensemble E.
Alors x est **idempotent** si, et seulement si, $x \star x = x$.

Sommaire

É.Galois (1811-1832)

Définition 15 (Groupe)

Un groupe est un monoïde tel que tout élément est inversible.

Définition 16 (Sous-groupe)

$(H, \star)$ est un sous-groupe de $(G, \star)$ si, et seulement si, H est une partie de G et $(H, \star)$ est un groupe.

Définition 15 (Groupe)

Un groupe est un monoïde tel que tout élément est inversible.

Définition 16 (Sous-groupe)

$\langle H, \star \rangle$ est un sous-groupe de $\langle G, \star \rangle$ si, et seulement si, H est une partie de G et $\langle H, \star \rangle$ est un groupe.

Définition 15 (Groupe)

Un groupe est un monoïde tel que tout élément est inversible.

Définition 16 (Sous-groupe)

$\langle H, \star \rangle$ est un sous-groupe de $\langle G, \star \rangle$ si, et seulement si, H est une partie de G et $\langle H, \star \rangle$ est un groupe.

Sommaire

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 17 (Anneau)

Soit A un ensemble muni de deux lois $\boxdot$ et $\boxplus$.
On dit que $\langle A, \boxplus, \boxdot \rangle$ est un anneau si, et seulement si :

- $\langle A, \boxplus \rangle$ est un groupe commutatif ;
- $\boxdot$ est associative ;
- $\boxdot$ est distributive sur $\boxplus$.

Si $\boxdot$ est commutative, alors l'anneau est dit *commutatif*.
Si $\boxdot$ admet un élément neutre, l'anneau est dit *unitaire*.
Les éléments d'un anneau unitaire qui admettent un symétrique par $\boxdot$ sont dits *inversibles*. On note A^* l'ensemble des éléments inversibles de A.

Définition 18 (Corps)

Soit $\mathbb{K}$ un ensemble muni de deux LCI $\boxplus$ et $\boxdot$. Le triplet $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ possède une structure de **corps** si, et seulement si,

- $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ a une structure d'anneau unitaire ;
- $\langle \mathbb{K} \setminus \{e_{\boxplus}\}, \boxdot \rangle$ a une structure de groupe.

Combien d'éléments a au minimum un corps ?

Définition 18 (Corps)

Soit $\mathbb{K}$ un ensemble muni de deux LCI $\boxplus$ et $\boxdot$. Le triplet $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ possède une structure de **corps** si, et seulement si,

- $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ a une structure d'anneau unitaire ;
- $\langle \mathbb{K} \setminus \{e_{\boxplus}\}, \boxdot \rangle$ a une structure de groupe.

Combien d'éléments a au minimum un corps ?

Définition 18 (Corps)

Soit $\mathbb{K}$ un ensemble muni de deux LCI $\boxplus$ et $\boxdot$. Le triplet $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ possède une structure de **corps** si, et seulement si,

- $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ a une structure d'anneau unitaire ;
- $\langle \mathbb{K} \setminus \{e_{\boxplus}\}, \boxdot \rangle$ a une structure de groupe.

Combien d'éléments a au minimum un corps ?

Définition 18 (Corps)

Soit $\mathbb{K}$ un ensemble muni de deux LCI $\boxplus$ et $\boxdot$. Le triplet $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ possède une structure de **corps** si, et seulement si,

- $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ a une structure d'anneau unitaire ;
- $\langle \mathbb{K} \setminus \{e_{\boxplus}\}, \boxdot \rangle$ a une structure de groupe.

Combien d'éléments a au minimum un corps ?

Définition 18 (Corps)

Soit $\mathbb{K}$ un ensemble muni de deux LCI $\boxplus$ et $\boxdot$. Le triplet $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ possède une structure de **corps** si, et seulement si,

- $\langle \mathbb{K}, \boxplus, \boxdot \rangle$ a une structure d'anneau unitaire ;
- $\langle \mathbb{K} \setminus \{e_{\boxplus}\}, \boxdot \rangle$ a une structure de groupe.

Combien d'éléments a au minimum un corps ?

```
class  (Eq a, Show a) => Num a  where
    (+), (-), (*)  :: a -> a -> a
    negate         :: a -> a
    abs, signum    :: a -> a
    fromInteger    :: Integer -> a
```

À quelle structure algébrique peut-on associer la classe `Num` ?

```
class  (Eq a, Show a) => Num a  where
    (+), (-), (*)  :: a -> a -> a
    negate         :: a -> a
    abs, signum    :: a -> a
    fromInteger    :: Integer -> a
```

À quelle structure algébrique peut-on associer la classe `Num` ?

```
class  (Num a) => Fractional a  where
    (/)          :: a -> a -> a
    recip        :: a -> a
    fromRational :: Rational -> a
```

À quelle structure algébrique peut-on associer la classe `Frational` ?

```
class  (Num a) => Fractional a  where
    (/)          :: a -> a -> a
    recip        :: a -> a
    fromRational :: Rational -> a
```

À quelle structure algébrique peut-on associer la classe `Frational` ?

Sommaire

John Backus (1924 - 2007)

Much of my work has come from being lazy

John Backus (1924 - 2007)

Much of my work has come from being lazy

Spécifier

SPÉCIFIER la fonction : c'est-à-dire parfois (on peut effectivement s'en passer au besoin) la nommer, donner son *type*, i.e. son domaine et son codomaine : on parle de *signature de type* et on explique ce qu'elle fait.

```
double :: Int -> Int
-- calcule le double d'un entier : le résultat est un entier
```

Spécifier

SPÉCIFIER la fonction : c'est-à-dire parfois (on peut effectivement s'en passer au besoin) la nommer, donner son *type*, i.e. son domaine et son codomaine : on parle de *signature de type* et on explique ce qu'elle fait.

```
double :: Int -> Int
-- calcule le double d'un entier : le résultat est un entier
```

Réaliser

RÉALISER la fonction c'est associer à la fonction spécifiée une expression. Pour cela, on utilise des *paramètres formels* qui font *abstraction* des valeurs particulières qui leur seront ensuite *substituées*.

```
double n = 2 * n
```

Réaliser

RÉALISER la fonction c'est associer à la fonction spécifiée une expression. Pour cela, on utilise des *paramètres formels* qui font *abstraction* des valeurs particulières qui leur seront ensuite *substituées*.

```
double n = 2 * n
```

Utiliser

UTILISER a fonction dans une expression en lui donnant des *paramètres effectifs* (arguments), c'est-à-dire liés à l'application particulière de la fonction :

```
*Main> (double 3) + (double 7)
20
```

Utiliser

UTILISER a fonction dans une expression en lui donnant des *paramètres effectifs* (arguments), c'est-à-dire liés à l'application particulière de la fonction :

```
*Main> (double 3) + (double 7)
20
```

Haskell Curry (1900 - 1982)


KEEP
CALM
AND
CURRY
ON

Une **fonction curryfiée** est une fonction de plusieurs variables transformée en une fonction d'une seule variable qui renvoie une fonction ayant une variable de moins...

Cela permet de créer des fonctions partielles.

Une **fonction curryfiée** est une fonction de plusieurs variables transformée en une fonction d'une seule variable qui renvoie une fonction ayant une variable de moins...
Cela permet de créer des fonctions partielles.

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y        =  x + y
```

Associativité

```
f = plus 3
```

Que vaut f(5) ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut f(5) ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut f(5) ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut f(5) ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut `f(5)` ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut `f(5)` ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

```
plus x y = x + y
```

```
plus :: Integer -> Integer -> Integer
plus    x          y       =  x + y
```

Associativité

```
f = plus 3
```

Que vaut `f(5)` ?

```
*Main> f 5
8
```

```
*Main> :t f
f :: Integer -> Integer
```

$\mathcal{F}(D, A)$

$\texttt{plus} \in \mathcal{F}(\mathbb{N}, \mathcal{F}(\mathbb{N}, \mathbb{N}))$

$\mathcal{F}(D, A)$
$\texttt{plus} \in \mathcal{F}(\mathbb{N}, \mathcal{F}(\mathbb{N}, \mathbb{N}))$

Exemple 19

Décrire une fonction qui étant donnés quatre flottants leur associe la moyenne des deux nombres parmi les quatre qui ne sont ni le plus grand ni le plus petit. Par exemple, la moyenne olympique de 10, 8, 12 et 14 est 11 et celle de 12, 12, 12 et 12 est 12.

Spécification

```
moyenne_olympique4 :: Float -> Float -> Float -> Float -> Float
-- renvoie la moyenne olympique de 4 flottants sous forme d'un
    flottant
```

Réalisation

Nous allons par exemple additionner les quatre nombres, retirer le plus grand et le plus petit puis diviser par 2.

```
*Main> :t min
min :: Ord a => a -> a -> a
```

```
*Main> min 'a' 'b'
'a'
*Main> min 5 3
3
*Main> min "Tralala" "Pouet Pouet"
"Pouet Pouet"
*Main> min True False
False
```

Réalisation

Nous allons par exemple additionner les quatre nombres, retirer le plus grand et le plus petit puis diviser par 2.

```
*Main> :t min
min :: Ord a => a -> a -> a
```

```
*Main> min 'a' 'b'
'a'
*Main> min 5 3
3
*Main> min "Tralala" "Pouet Pouet"
"Pouet Pouet"
*Main> min True False
False
```

Réalisation

Nous allons par exemple additionner les quatre nombres, retirer le plus grand et le plus petit puis diviser par 2.

```
*Main> :t min
min :: Ord a => a -> a -> a
```

```
*Main> min 'a' 'b'
'a'
*Main> min 5 3
3
*Main> min "Tralala" "Pouet Pouet"
"Pouet Pouet"
*Main> min True False
False
```

Réalisation

```
moyenne_olympique4 a b c d =
  (s - mini - maxi) / 2 -- l'expression correspondant à la méthode
      choisie
  where s = a + b + c + d -- la somme des 4 nombres
        mini = min a (min  b (min c d) ) -- le plus petit des 4
        maxi = max a (max  b (max c d) ) -- le plus grand des 4
```

```
*Main> moyenne_olympique4 10 8 12 14
11.0
```

Réalisation

```
moyenne_olympique4 a b c d =
  (s - mini - maxi) / 2 -- l'expression correspondant à la méthode
      choisie
  where s = a + b + c + d -- la somme des 4 nombres
        mini = min a (min  b (min c d) ) -- le plus petit des 4
        maxi = max a (max  b (max c d) ) -- le plus grand des 4
```

```
*Main> moyenne_olympique4 10 8 12 14
11.0
```

Exemple 20

Décrire une fonction qui étant donnée une liste d'au moins quatre nombres leur associe la moyenne des nombres parmi ceux de la liste qui ne sont ni le plus grand ni le plus petit. Par exemple, la moyenne olympique de 15, 7, 10, 8, 12 et 14 est 11.

Min et Max polymorphes

```
minListe :: (Ord a) => [a] -> a
-- renvoie le mini d'une liste d'éléments ordonnables
```

```
maxListe :: (Ord a) => [a] -> a
-- renvoie le maxi d'une liste d'éléments ordonnables
```

```
extr_liste :: (Ord a) => (a -> a -> a) -> [a] -> a
-- renvoie le mini ou le maxi (on choisit en mettant la nature de l'
    extremum en paramètre) d'une liste d'éléments ordonnables
```

Min et Max polymorphes

```
minListe :: (Ord a) => [a] -> a
-- renvoie le mini d'une liste d'éléments ordonnables
```

```
maxListe :: (Ord a) => [a] -> a
-- renvoie le maxi d'une liste d'éléments ordonnables
```

```
extr_liste :: (Ord a) => (a -> a -> a) -> [a] -> a
-- renvoie le mini ou le maxi (on choisit en mettant la nature de l'
    extremum en paramètre) d'une liste d'éléments ordonnables
```

Min et Max polymorphes

```
minListe :: (Ord a) => [a] -> a
-- renvoie le mini d'une liste d'éléments ordonnables
```

```
maxListe :: (Ord a) => [a] -> a
-- renvoie le maxi d'une liste d'éléments ordonnables
```

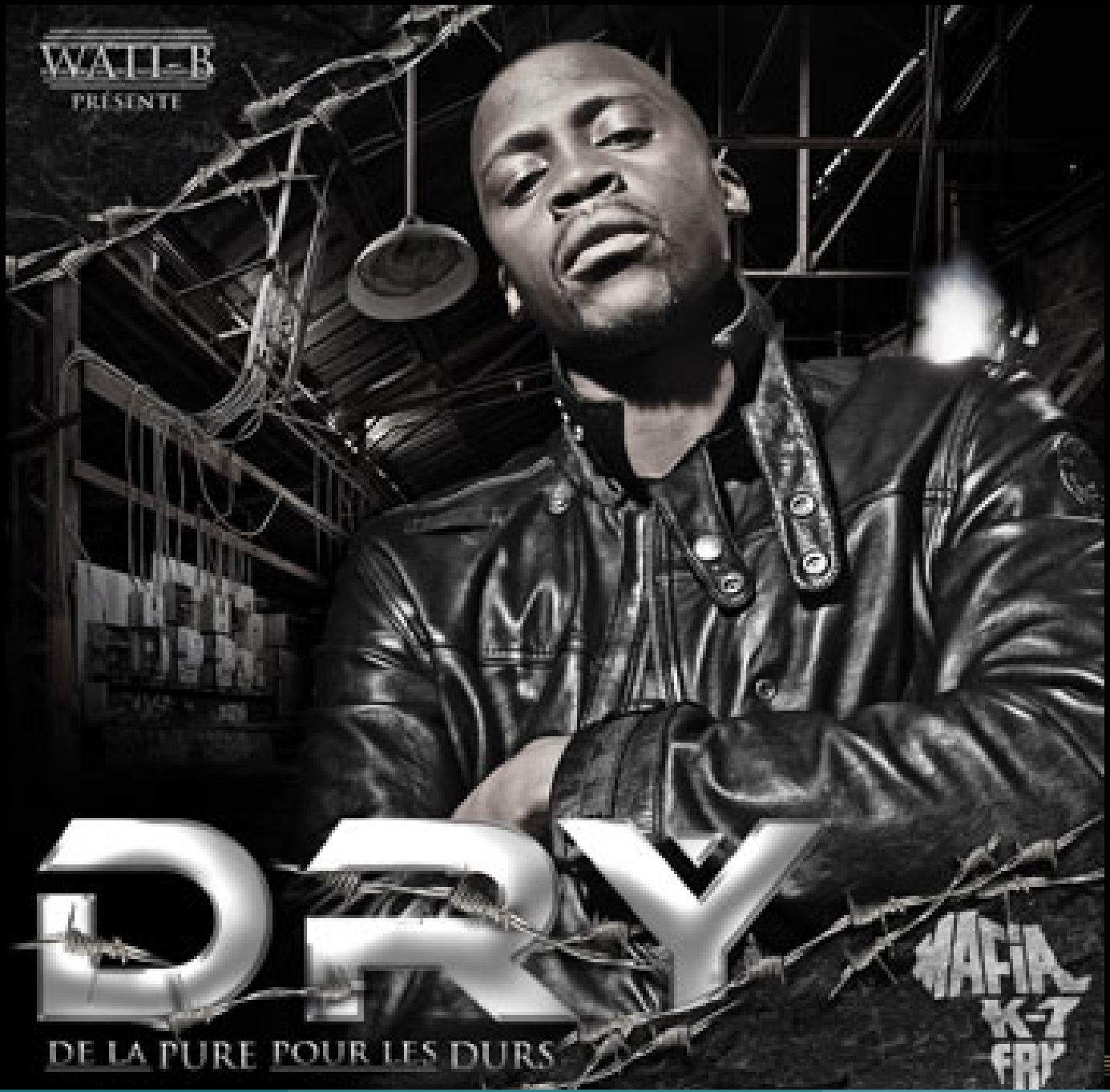
```
extr_liste :: (Ord a) => (a -> a -> a) -> [a] -> a
-- renvoie le mini ou le maxi (on choisit en mettant la nature de l'
    extremum en paramètre) d'une liste d'éléments ordonnables
```

```
m_o4 :: (Fractional t, Ord t) => t -> t -> t -> t -> t
m_o4                  a    b    c    d =
  (s - mini - maxi) / 2
  where {
    s    = a + b + c + d ;
    mini = min d (min c (min a b)) ;
    maxi = max d (max c (max a b))
        }
```

```
som_liste :: (Num typ) => [typ]                          -> typ
som_liste                 []                             =
    error "Ta liste est vide !!"
som_liste                 (tete_de_liste : [])           =
    tete_de_liste
som_liste                 (tete_de_liste : liste_décapitée)  =
    (+) tete_de_liste (som_liste liste_décapitée)
```

```
minListe :: ( Ord t) => [t]                    -> t
minListe                []                     =
    error "Ta liste est vide !!"
minListe                (tete : [])            =
    tete
minListe                (tete : liste_décapitée) =
    min tete (minListe liste_décapitée)
```

```
maxListe                []                      =
    error "Ta liste est vide !!"
maxListe                (tete : [])             =
    tete
maxListe                (tete : liste_décapitée) =
    max tete (maxListe liste_décapitée)
```


WALI-B
PRÉSENTE
DRY
DE LA PURE POUR LES DURS
MAFIA
K-7
FRY

```
op_liste = foldl1
```

fold comme...

l comme...

```
op_liste = foldl1
```

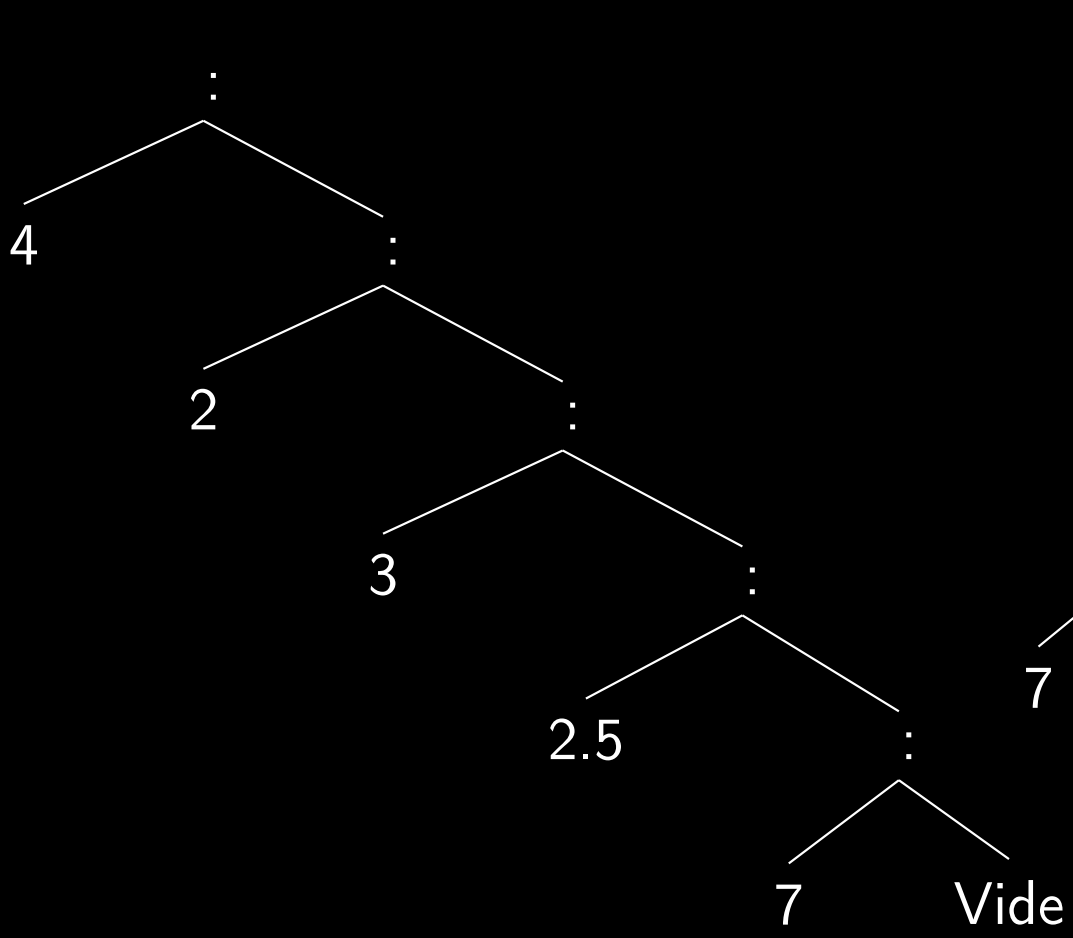
fold comme...

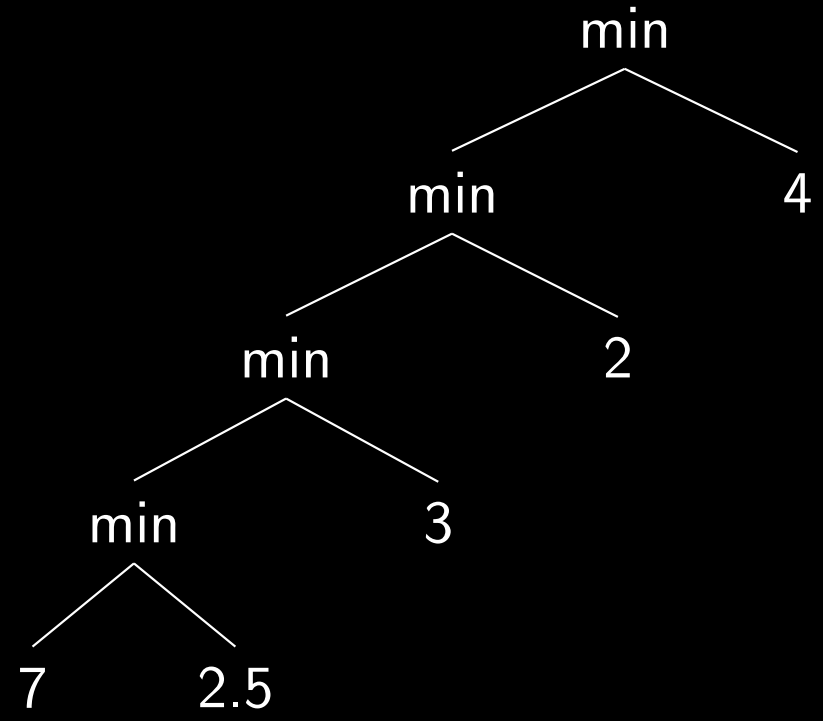
l comme...

```
op_liste = foldl1
```

`fold` comme...

`l` comme...


:
4
:
2
:
3
:
2.5
:
7
Vide


min
min
4
min
2
min
3
7
2.5

```
*Main> :t opListe
opListe :: (a -> a -> a) -> [a] -> a

*Main> :t opListe min
opListe min :: Ord a => [a] -> a

*Main> opListe min [4, 2, 3, 2.5, 7]
2.0

*Main> opListe max [4, 2, 3, 2.5, 7]
7.0

*Main> opListe (+) [4, 2, 3, 2.5, 7]
18.5

*Main> opListe (*) [4, 2, 3, 2.5, 7]
420.0
```

Longueur d'une liste ?

```
*Main> length [2,3,6,7]
4
```

Longueur d'une liste ?

```
*Main> length [2,3,6,7]
4
```

```
*Main> (opListe (+) [2,3,6,7]) / (length [2,3,6,7])
```

```
<interactive>:27:25:
    No instance for (Fractional Int)
      arising from a use of '/'
    Possible fix: add an instance declaration for (Fractional Int)
    In the expression:
      (opListe (+) [2, 3, 6, 7]) / (length [2, 3, 6, 7])
    In an equation for 'it':
        it = (opListe (+) [2, 3, 6, ....]) / (length [2, 3, 6,
              ....])
```

```
*Main> (opListe (+) [2,3,6,7]) / (length [2,3,6,7])
```

```
<interactive>:27:25:
    No instance for (Fractional Int)
      arising from a use of ‘/’
    Possible fix: add an instance declaration for (Fractional Int)
    In the expression:
      (opListe (+) [2, 3, 6, 7]) / (length [2, 3, 6, 7])
    In an equation for ‘it’:
        it = (opListe (+) [2, 3, 6, ....]) / (length [2, 3, 6,
              ....])
```

```
*Main> :t length
length :: [a] -> Int

*Main> :t (/)
(/) :: Fractional a => a -> a -> a
```

```
*Main> :t foldl1
foldl1 :: (a -> a -> a) -> [a] -> a
*Main> :t foldl
foldl :: (a -> b -> a) -> a -> [b] -> a
```

```
longListeFrac :: (Fractional nb) => [patates] -> nb
longListeFrac                       xs       =  foldl (\acc patate
    -> 1 + acc) 0 xs
```

```
*Main> longListeFrac ['a'..'z']
26.0
```

```
longListeFrac :: (Fractional nb) => [patates] -> nb
longListeFrac                       xs       =  foldl (\acc patate
    -> 1 + acc) 0 xs
```

```
*Main> longListeFrac ['a'..'z']
26.0
```

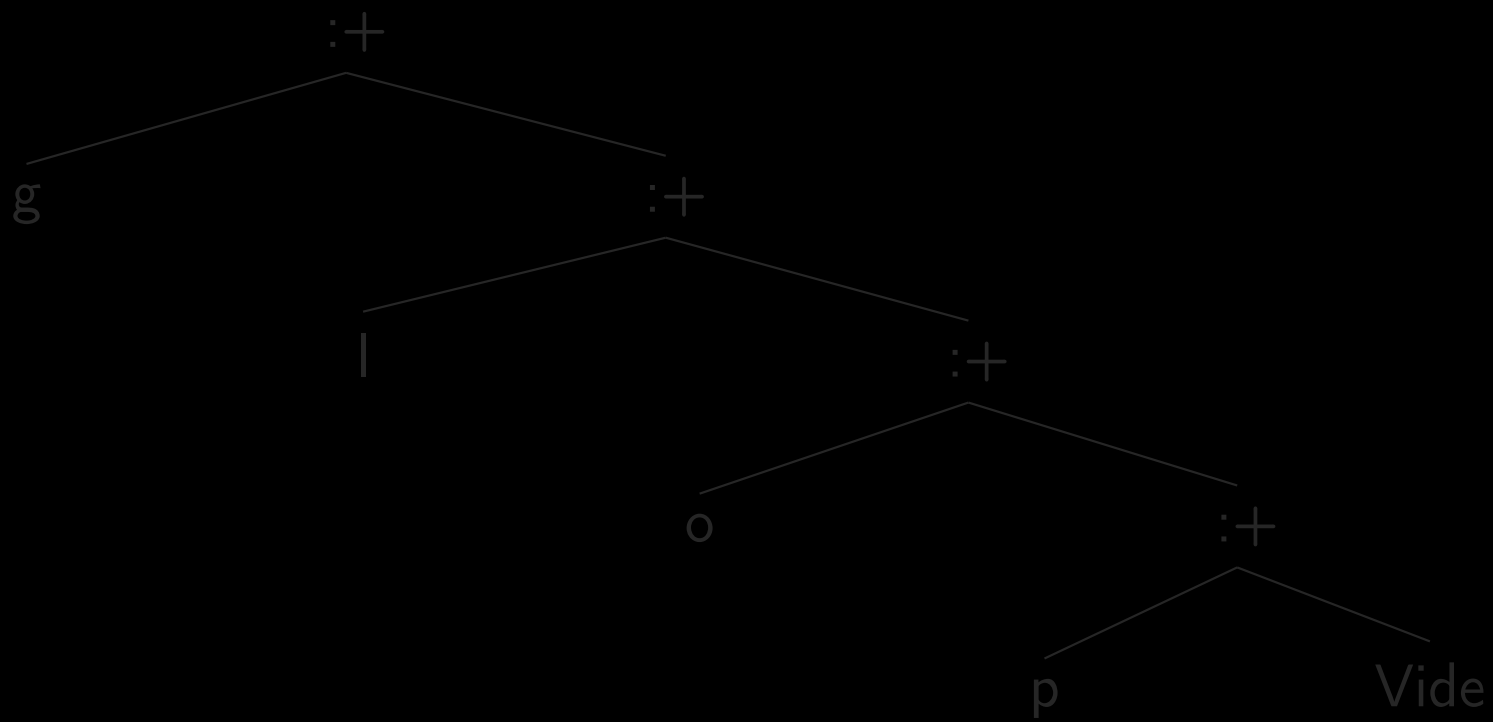
Alternative récursive :

```
LongListeFrac :: Fractional nb => [patates] -> nb
LongListeFrac                      []        =  0
LongListeFrac      (tete : liste_décapitée) =  1 + (LongListeFrac
    liste_décapitée)
```

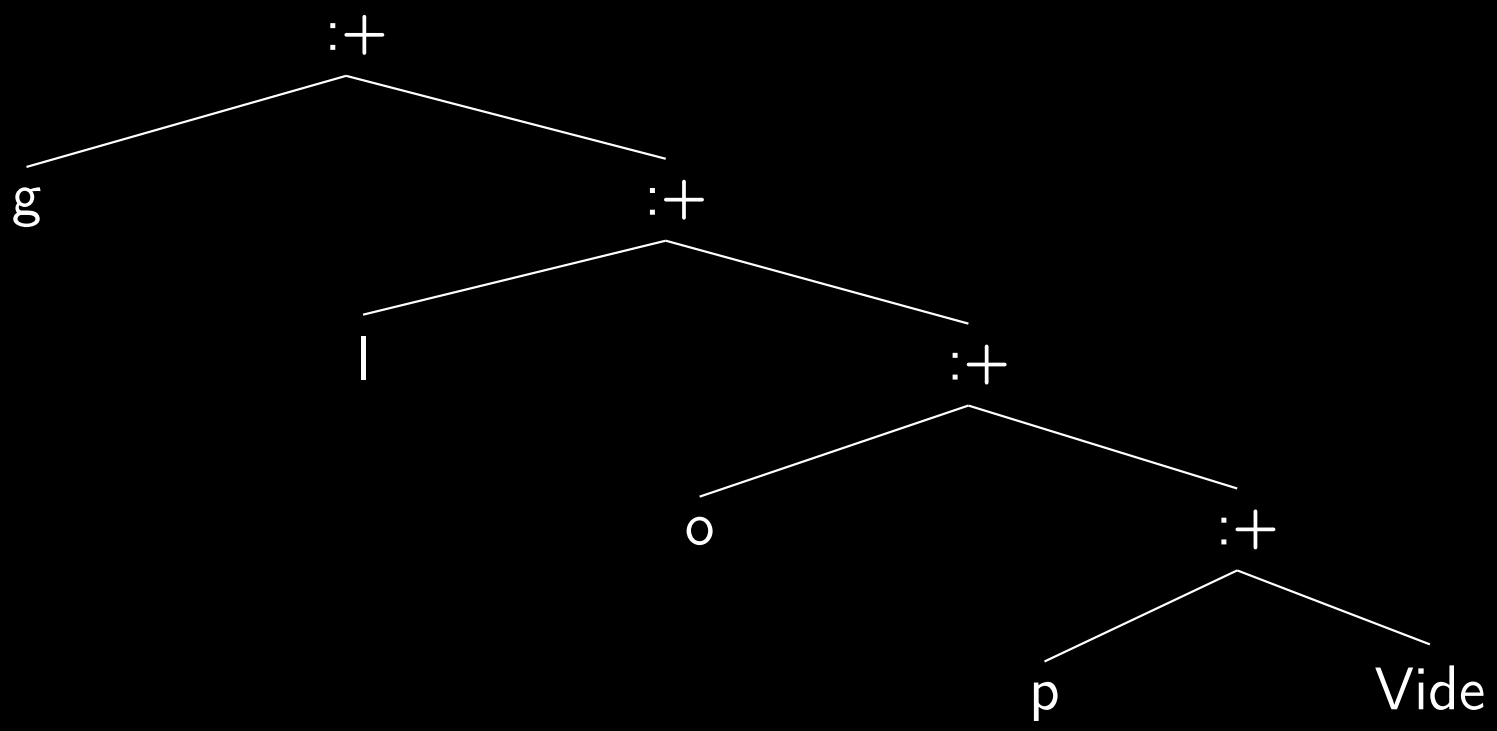
```
m_o :: (Ord t, Fractional t) => [t] -> t
m_o     liste_notes      =
                        if nb_notes <= 0 then error "Liste trop
                            courte !"
                        else
                          (som_notes - note_mini - note_maxi) /
                              nb_notes
  where {
    som_notes =  som_liste  liste_notes ;
    note_mini =  minListe  liste_notes ;
    note_maxi =  maxListe  liste_notes ;
    nb_notes  =  (LongListeFrac liste_notes) - 2
    }
```

Qu'est-ce qu'un mot ?

« glop »



« glop »



```haskell
data Mot =
  Vide
  | Char :+ Mot
```

```haskell
data Mot =
  Vide
  | Char :+ Mot
  deriving (Show)
```

```haskell
data Mot =
  Vide
  | Char :+ Mot
  deriving (Show, Eq)
```

```
*Main> let m = 'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide

<interactive>:44:9:
    Couldn't match expected type 'Char' with actual type 'Mot'
    In the first argument of '(:+)', namely 'g' :+ 'l' :+ 'o' :+ 'p'
    In the expression: 'g' :+ 'l' :+ 'o' :+ 'p' :+ Vide
    In an equation for 'm': m = 'g' :+ 'l' :+ 'o' :+ 'p' :+ Vide
```

```
*Main> let m = 'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide

<interactive>:44:9:
    Couldn't match expected type `Char' with actual type `Mot'
    In the first argument of `(:+)', namely 'g' :+ 'l' :+ 'o' :+ 'p'
    In the expression: 'g' :+ 'l' :+ 'o' :+ 'p' :+ Vide
    In an equation for `m': m = 'g' :+ 'l' :+ 'o' :+ 'p' :+ Vide
```

```haskell
infixr :+

data Mot =
  Vide
  | Char :+ Mot
  deriving (Show, Eq)
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
infixr :+

data Mot =
  Vide
  | Char :+ Mot
  deriving (Show, Eq)
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
*Main> :t m
m :: Mot
```

```
*Main> let m = 'g' :+ 'l'  :+ 'o' :+ 'p' :+ Vide
*Main> m
'g' :+ ('l' :+ ('o' :+ ('p' :+ Vide)))
```

```
*Main> :t m
m :: Mot
```

Sélecteurs

```
tete :: Mot   -> Char
-- renvoie le premier caractère d'un mot avec un filtrage par motif
tete    Vide  =  error "Mot vide !"
tete (t :+ q) =  t
```

```
*Main> tete m
'g'
```

Et la queue ?

Sélecteurs

```
tete :: Mot   -> Char
-- renvoie le premier caractère d'un mot avec un filtrage par motif
tete    Vide  =  error "Mot vide !"
tete (t :+ q) =  t
```

```
*Main> tete m
'g'
```

Et la queue ?

Sélecteurs

```
tete :: Mot   -> Char
-- renvoie le premier caractère d'un mot avec un filtrage par motif
tete    Vide  =  error "Mot vide !"
tete (t :+ q) =  t
```

```
*Main> tete m
'g'
```

Et la queue ?

Testeurs

```
estVide :: Mot -> Bool
estVide    m   =  m == Vide
```

$$\neg(\text{estVide } m) \leftrightarrow (m = (\text{tete } m) :+ (\text{queue } m))$$

Testeurs

```
estVide :: Mot -> Bool
estVide    m   =  m == Vide
```

$$\neg(\text{estVide } m) \leftrightarrow (m = (\text{tete } m) :+ (\text{queue } m))$$

Construire une fonction définie sur un type récursif

On voudrait compter le nombre de « a » dans un mot.

```
nba :: Mot -> Int
-- calcule le nombre de 'a' dans un mot
```

Construire une fonction définie sur un type récursif

On voudrait compter le nombre de « a » dans un mot.

```
nba :: Mot -> Int
-- calcule le nombre de 'a' dans un mot
```

- Si le mot est vide, alors son nombre de 'a' est 0.
- Sinon, le mot est construit par `t :+ q`.
  Si `t` = 'a', alors `nba mot` = 1 + `nba q`, sinon, `nba mot` = `nba q`.

```
nba Vide      = 0
nba (t :+ q) = (nba q) + (if t == 'a' then 1 else 0)
```

- Si le mot est vide, alors son nombre de 'a' est 0.
- Sinon, le mot est construit par `t :+ q`.
  Si `t` = 'a', alors `nba mot` = 1 + `nba q`, sinon, `nba mot` = `nba q`.

```
nba Vide      = 0
nba (t :+ q) = (nba q) + (if t == 'a' then 1 else 0)
```

Filtrage par motif

```
nba2 Vide = 0
nba2 (t :+ q)
  | t == 'a' = (nba2 q) + 1
  |otherwise =  nba2 q
```

Case

```
nba3 mot = case mot of
                Vide     -> 0
                (t :+ q) -> (nba3 q) + (if t == 'a' then 1 else 0)
```

Pliage

```
nba4 :: Mot -> Int
-- calcule le nombre de 'a' dans un mot par pliage
nba4 mot =
  pliage (\accu t -> accu + (if t == 'a' then 1 else 0)) 0 mot
```

Pliage

```
pliage :: (a -> Char -> a) -> a -> Mot -> a
-- adapte foldl aux mots
pliage fonc ini Vide     = ini
pliage fonc ini (t :+ q) = pliage fonc (fonc ini t) q
```

Applique

Mettre en majuscule.

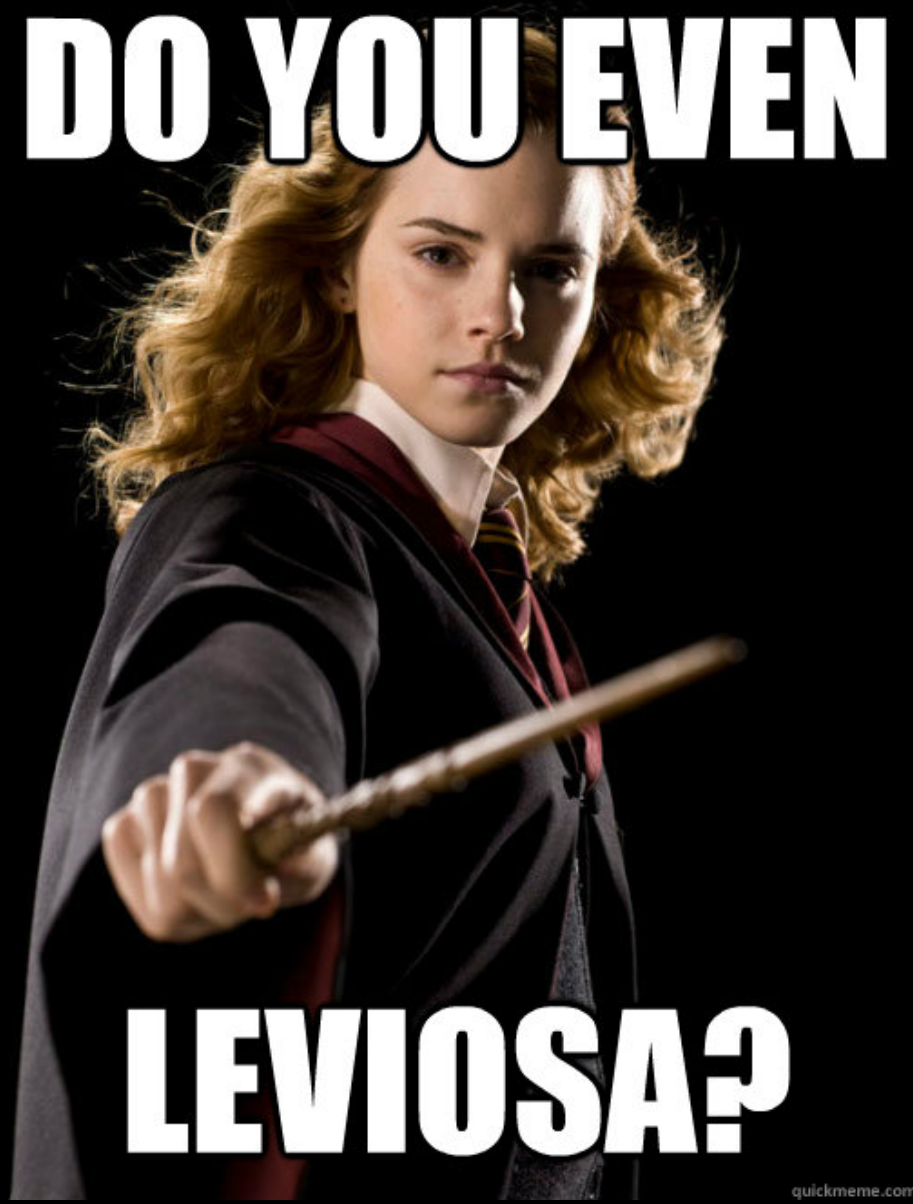
```
*Main> import Data.Char
*Main Data.Char> toUpper 'g'
'G'
```

Applique

Mettre en majuscule.

```
*Main> import Data.Char
*Main Data.Char> toUpper 'g'
'G'
```

glop -> GLOP ?


DO YOU EVEN
LEVIOSA?
quickmeme.com

```
applique :: (Char -> Char) -> Mot     -> Mot
applique    f                 Vide    =  Vide
applique    f                (t :+ q) =  (f t) :+ (applique f q)
```

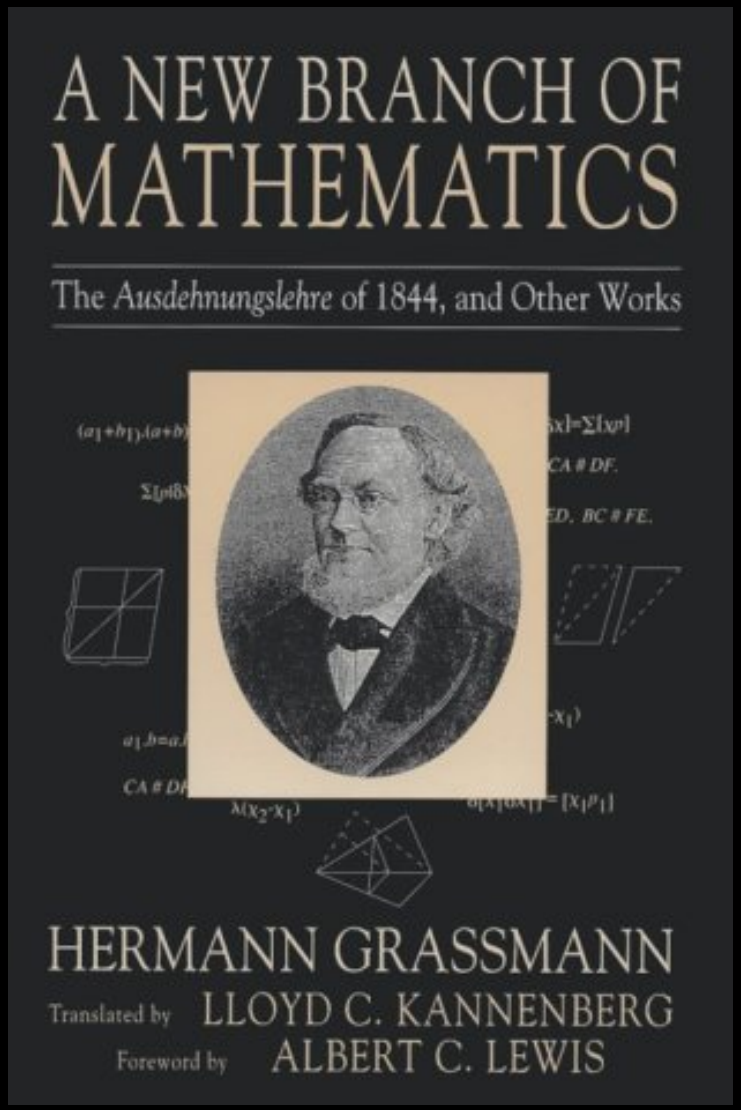
```
*Main> applique Data.Char.toUpper m
'G' :+ ('L' :+ ('O' :+ ('P' :+ Vide)))
```

```
applique :: (Char -> Char) -> Mot     -> Mot
applique    f                 Vide    =  Vide
applique    f                (t :+ q) =  (f t) :+ (applique f q)
```

```
*Main> applique Data.Char.toUpper m
'G' :+ ('L' :+ ('O' :+ ('P' :+ Vide)))
```


FOR SOME DOCTORS,
THERE ARE NO BOUNDARIES.
OFF THE MAP

Sommaire



Hermann Graßmann (1809-1877)

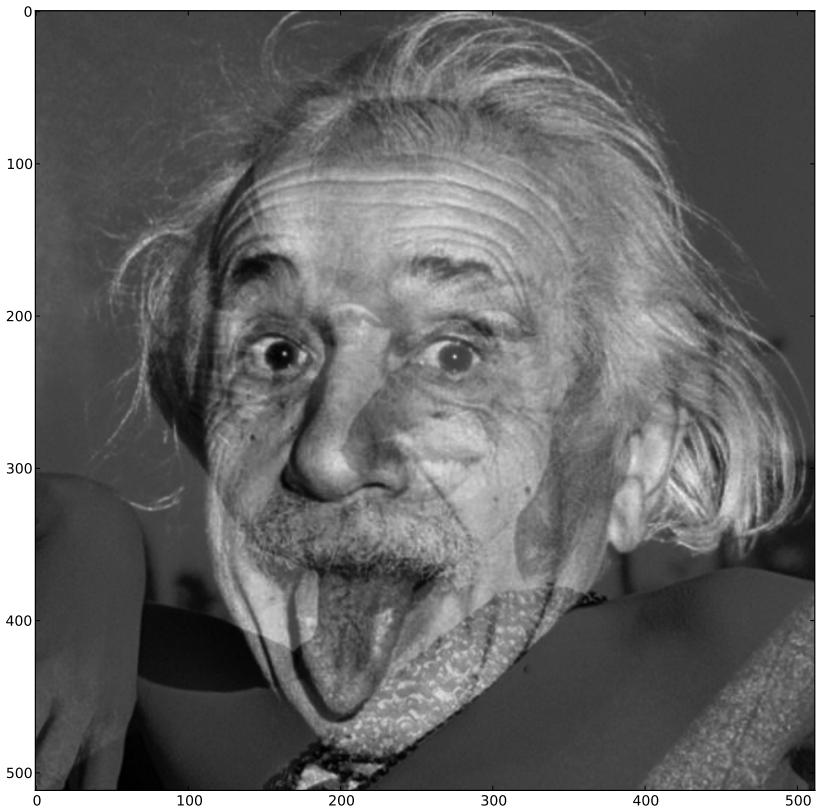
Stefan Banach (1892-1945)

$3 \times \text{Marylin} + \text{Albert}$

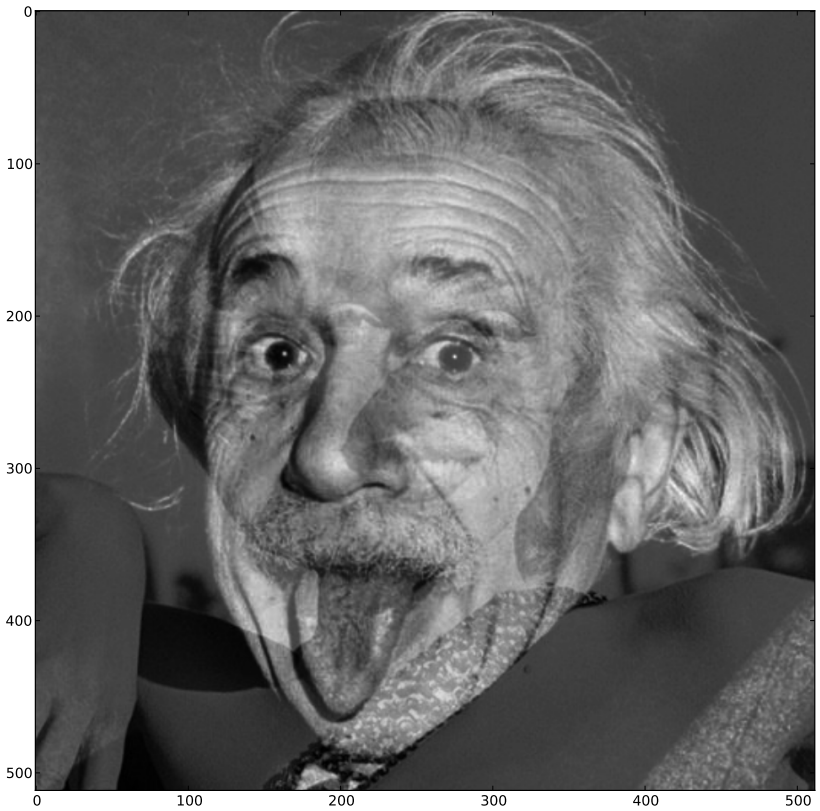
$3 \times \text{Marylin} + \text{Albert}$

Marylin + Albert

Marylin + Albert



Marylin + 3 × Albert



Marylin + 3 × Albert

Définition 21 ($\mathbb{K}$-espace vectoriel)

Un $\mathbb{K}$-espace vectoriel V sur un corps commutatif $\langle \mathbb{K}, \oplus, \odot \rangle$ est un groupe abélien $\langle V, \dagger \rangle$ muni d'une loi de composition *externe* $\cdot$ de $\mathbb{K} \otimes V$ dans V vérifiant les quatre axiomes suivant, λ et μ désignant des scalaires quelconques, $\boldsymbol{u}$ et $\boldsymbol{v}$ des vecteurs quelconques et $1_\odot$ l'élément neutre de la loi $\odot$ sur $\mathbb{K}$:

$(\lambda \oplus \mu) \cdot \boldsymbol{u} = \lambda \cdot \boldsymbol{u} \dagger \mu \cdot \boldsymbol{u}$ (distributivité vectorielle) ;

$\lambda \cdot (\boldsymbol{u} \dagger \boldsymbol{v}) = \lambda \cdot \boldsymbol{u} \dagger \lambda \cdot \boldsymbol{v}$ (distributivité scalaire) ;

$(\lambda \odot \mu) \cdot \boldsymbol{u} = \lambda \cdot (\mu \cdot \boldsymbol{u})$ (associativité) ;

$1_\odot \cdot \boldsymbol{u} = \boldsymbol{u}$ (axiome d'identité)

On dit que le corps $\mathbb{K}$ *opère* sur le groupe $\langle V, \dagger \rangle$.

```
imshow(3 * marilyn + einstein, cmap = cm.gray)
```

Définition 21 ($\mathbb{K}$-espace vectoriel)

Un $\mathbb{K}$-espace vectoriel V sur un corps commutatif $\langle \mathbb{K}, \oplus, \odot \rangle$ est un groupe abélien $\langle V, \dagger \rangle$ muni d'une loi de composition *externe* $\cdot$ de $\mathbb{K} \otimes V$ dans V vérifiant les quatre axiomes suivant, λ et μ désignant des scalaires quelconques, $\boldsymbol{u}$ et $\boldsymbol{v}$ des vecteurs quelconques et $1_\odot$ l'élément neutre de la loi $\odot$ sur $\mathbb{K}$:

$(\lambda \oplus \mu) \cdot \boldsymbol{u} = \lambda \cdot \boldsymbol{u} \dagger \mu \cdot \boldsymbol{u}$ (distributivité vectorielle) ;

$\lambda \cdot (\boldsymbol{u} \dagger \boldsymbol{v}) = \lambda \cdot \boldsymbol{u} \dagger \lambda \cdot \boldsymbol{v}$ (distributivité scalaire) ;

$(\lambda \odot \mu) \cdot \boldsymbol{u} = \lambda \cdot (\mu \cdot \boldsymbol{u})$ (associativité) ;

$1_\odot \cdot \boldsymbol{u} = \boldsymbol{u}$ (axiome d'identité)

On dit que le corps $\mathbb{K}$ *opère* sur le groupe $\langle V, \dagger \rangle$.

```
imshow(3 * marilyn + einstein, cmap = cm.gray)
```

$$\langle b_1, b_2, b_3\rangle \dagger \langle b'_1, b'_2, b'_3\rangle = \langle b_1 \oplus b'_1, b_2 \oplus b'_2, b_3 \oplus b'_3\rangle$$

$$(\cdot) : \begin{array}{ccc} \mathbb{F}_2 \otimes T & \longrightarrow & T \\ \langle \lambda, \langle b_1, b_2, b_3\rangle\rangle & \longmapsto & \langle \lambda \odot b_1, \lambda \odot b_2, \lambda \odot b_3\rangle \end{array}$$

$$\langle b_1, b_2, b_3\rangle \dagger \langle b'_1, b'_2, b'_3\rangle = \langle b_1 \oplus b'_1, b_2 \oplus b'_2, b_3 \oplus b'_3\rangle$$

$$\begin{array}{rccc} (\cdot) : & \mathbb{F}_2 \otimes T & \to & T \\ & \langle \lambda, \langle b_1, b_2, b_3\rangle\rangle & \mapsto & \langle \lambda \odot b_1, \lambda \odot b_2, \lambda \odot b_3\rangle \end{array}$$

```haskell
data BoolInt = O | I deriving(Show,Read,Eq)

type BitChaine = [BoolInt]

bPlus :: BoolInt -> BoolInt -> BoolInt
bPlus    b1          b2
  |b1 == b2                  = O
  |otherwise                 = I

bFois :: BoolInt -> BoolInt -> BoolInt
bFois    I           I       =  I
bFois    _           _       =  O

cPlus :: BitChaine -> BitChaine -> BitChaine
cPlus    c1            c2          =  zipWith bPlus c1 c2

prodExt :: BoolInt -> BitChaine -> BitChaine
prodExt    b           c           =  map (\x -> bFois b x) c
```

```
*Main> bPlus O I
I
*Main> cPlus [O,O,I] [I,O,I]
[I,O,O]
*Main> prodExt I [O,O,I]
[O,O,I]
*Main> prodExt O [O,O,I]
[O,O,O]
```

```
lesBits = [O,I]

lesTrios = [[x,y,z] | x <- lesBits, y <- lesBits, z <- lesBits]

and [prodExt (bPlus b1 b2) t == cPlus (prodExt b1 t) (prodExt b2 t)
    | b1 <- lesBits, b2 <- lesBits, t <- lesTrios ]
```

Définition 22 (Combinaison linéaire de vecteurs)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ un corps opérant sur le groupe $\langle V, \dagger \rangle$ pour former un espace vectoriel de produit externe $\bullet$.
Soit $\langle \boldsymbol{u_0}, \boldsymbol{u_1}, \cdots, \boldsymbol{u_p} \rangle$ une famille de vecteurs de V.
Un vecteur $\boldsymbol{v}$ de V est une combinaison linéaire de la famille des $\langle \boldsymbol{u_i} \rangle_{0 \leqslant i \leqslant p}$ si, et seulement si, il existe une famille $\langle \lambda_0, \lambda_1, \cdots, \lambda_p \rangle$ de scalaires de $\mathbb{K}$ vérifiant :

$$\boldsymbol{v} = \bigdagger_{i=0}^{i=p} \lambda_i \bullet \boldsymbol{u_i} = \lambda_0 \bullet \boldsymbol{u_0} \dagger \lambda_1 \bullet \boldsymbol{u_1} \dagger \cdots \dagger \lambda_p \bullet \boldsymbol{u_p}$$

L'ensemble de ces combinaisons linéaires est noté :

$$\mathcal{V}\text{ect}\{\boldsymbol{u_0}, \boldsymbol{u_1}, \cdots, \boldsymbol{u_p}\}$$

```
u = [O,I,I]
v = [I,O,I]
vect = [cPlus (prodExt b1 u) (prodExt b2 v) | b1 <- lesBits, b2 <-
    lesBits]
```

```
*Main> vect
[[O,O,O],[I,O,I],[O,I,I],[I,I,O]]
```

```
u = [O,I,I]
v = [I,O,I]
vect = [cPlus (prodExt b1 u) (prodExt b2 v) | b1 <- lesBits, b2 <-
    lesBits]
```

```
*Main> vect
[[O,O,O],[I,O,I],[O,I,I],[I,I,O]]
```

Définition 23 (Sev)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ un corps opérant sur le groupe $\langle V, \dagger \rangle$ pour former un espace vectoriel de produit externe $\bullet$.
Soit W un sous-ensemble de V.
Si, muni des mêmes opérations que V, W a une structure de $\mathbb{K}$-espace vectoriel, alors on dit que W est un **sous-espace vectoriel** de V.

Théorème 24 (Caractérisation (1) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $W \neq \emptyset$
- $(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \boldsymbol{u}, \boldsymbol{v} \rangle \in W^2 \rightarrow \boldsymbol{u} \dagger \boldsymbol{v} \in W)$
- $(\forall \lambda)(\forall \boldsymbol{u})(\langle \lambda, \boldsymbol{u} \rangle \in \mathbb{K} \otimes W \rightarrow \lambda \bullet \boldsymbol{u} \in W)$

Théorème 24 (Caractérisation (1) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $W \neq \emptyset$
- $(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \boldsymbol{u}, \boldsymbol{v} \rangle \in W^2 \rightarrow \boldsymbol{u} \dagger \boldsymbol{v} \in W)$
- $(\forall \lambda)(\forall \boldsymbol{u})(\langle \lambda, \boldsymbol{u} \rangle \in \mathbb{K} \otimes W \rightarrow \lambda \bullet \boldsymbol{u} \in W)$

Théorème 24 (Caractérisation (1) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $W \neq \emptyset$
- $(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \boldsymbol{u}, \boldsymbol{v} \rangle \in W^2 \rightarrow \boldsymbol{u} \dagger \boldsymbol{v} \in W)$
- $(\forall \lambda)(\forall \boldsymbol{u})(\langle \lambda, \boldsymbol{u} \rangle \in \mathbb{K} \otimes W \rightarrow \lambda \bullet \boldsymbol{u} \in W)$

Théorème 24 (Caractérisation (1) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $W \neq \emptyset$
- $(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \boldsymbol{u}, \boldsymbol{v} \rangle \in W^2 \rightarrow \boldsymbol{u} \dagger \boldsymbol{v} \in W)$
- $(\forall \lambda)(\forall \boldsymbol{u})(\langle \lambda, \boldsymbol{u} \rangle \in \mathbb{K} \otimes W \rightarrow \lambda \bullet \boldsymbol{u} \in W)$

Théorème 25 (Caractérisation (2) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $\mathbf{0} \in W$
- $(\forall \lambda)(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \lambda, \boldsymbol{u}, \boldsymbol{v} \rangle \in \mathbb{K} \otimes W \otimes W \rightarrow \boldsymbol{u} \dagger \lambda \bullet \boldsymbol{v} \in W)$

Théorème 25 (Caractérisation (2) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $\mathbf{0} \in W$
- $(\forall \lambda)(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \lambda, \boldsymbol{u}, \boldsymbol{v} \rangle \in \mathbb{K} \otimes W \otimes W \rightarrow \boldsymbol{u} \dagger \lambda \bullet \boldsymbol{v} \in W)$

Théorème 25 (Caractérisation (2) des sous-espaces vectoriels)

Soit $\langle \mathbb{K}, \oplus, \odot \rangle$ *un corps opérant sur le groupe* $\langle V, \dagger \rangle$ *pour former un espace vectoriel de produit externe* $\bullet$.
Soit W *un sous-ensemble de* V.
W *est un sous-espace vectoriel (sev) de* V *si, et seulement si :*

- $\mathbf{0} \in W$
- $(\forall \lambda)(\forall \boldsymbol{u})(\forall \boldsymbol{v})(\langle \lambda, \boldsymbol{u}, \boldsymbol{v} \rangle \in \mathbb{K} \otimes W \otimes W \rightarrow \boldsymbol{u} \dagger \lambda \bullet \boldsymbol{v} \in W)$

Sommaire

方程


DEUTSCHE BUNDESBANK
Zehn Deutsche Mark
10
1777–1855 Carl Friedr. Gauß
DG5533672Z4

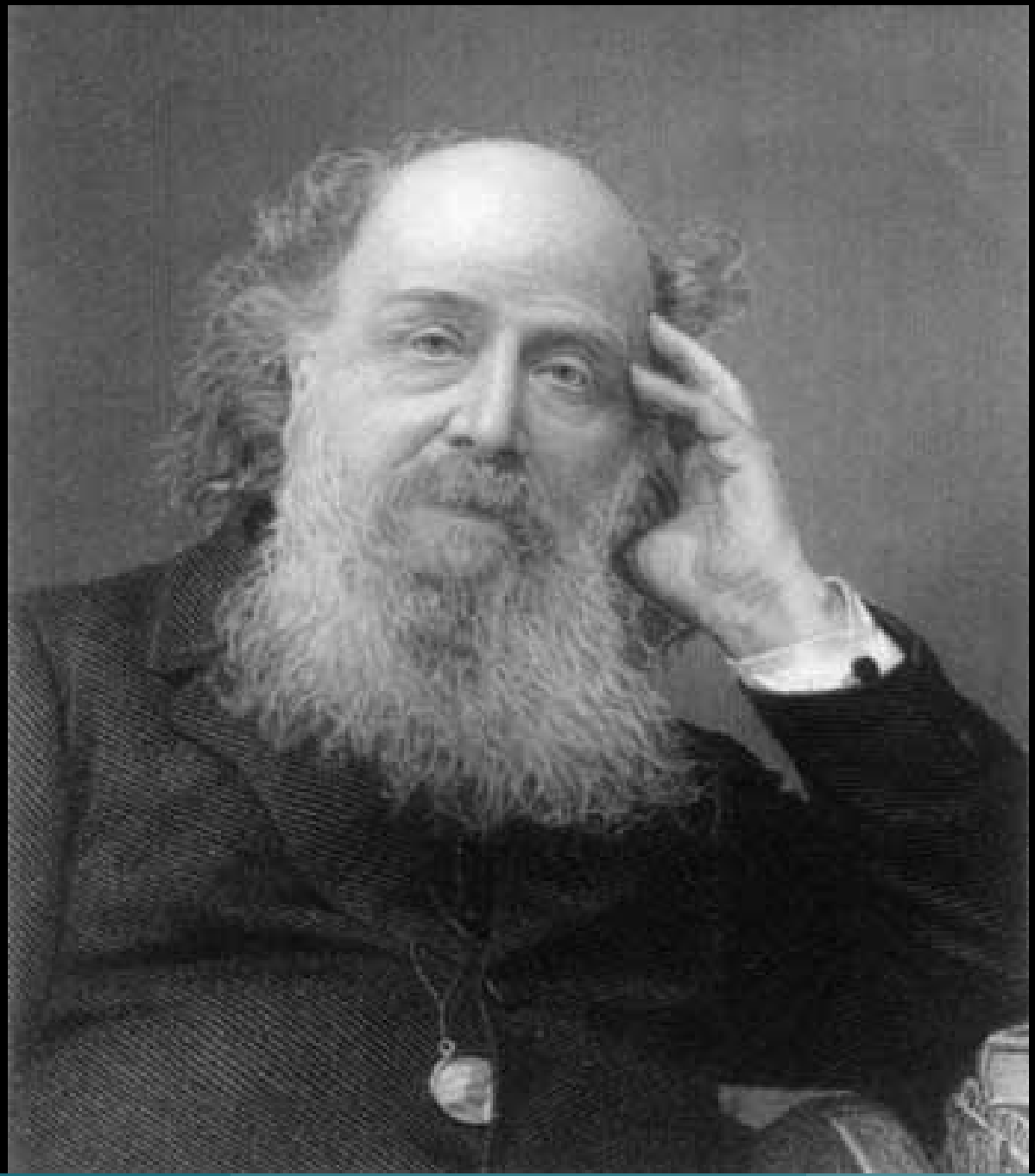
Wilhelm Jordan

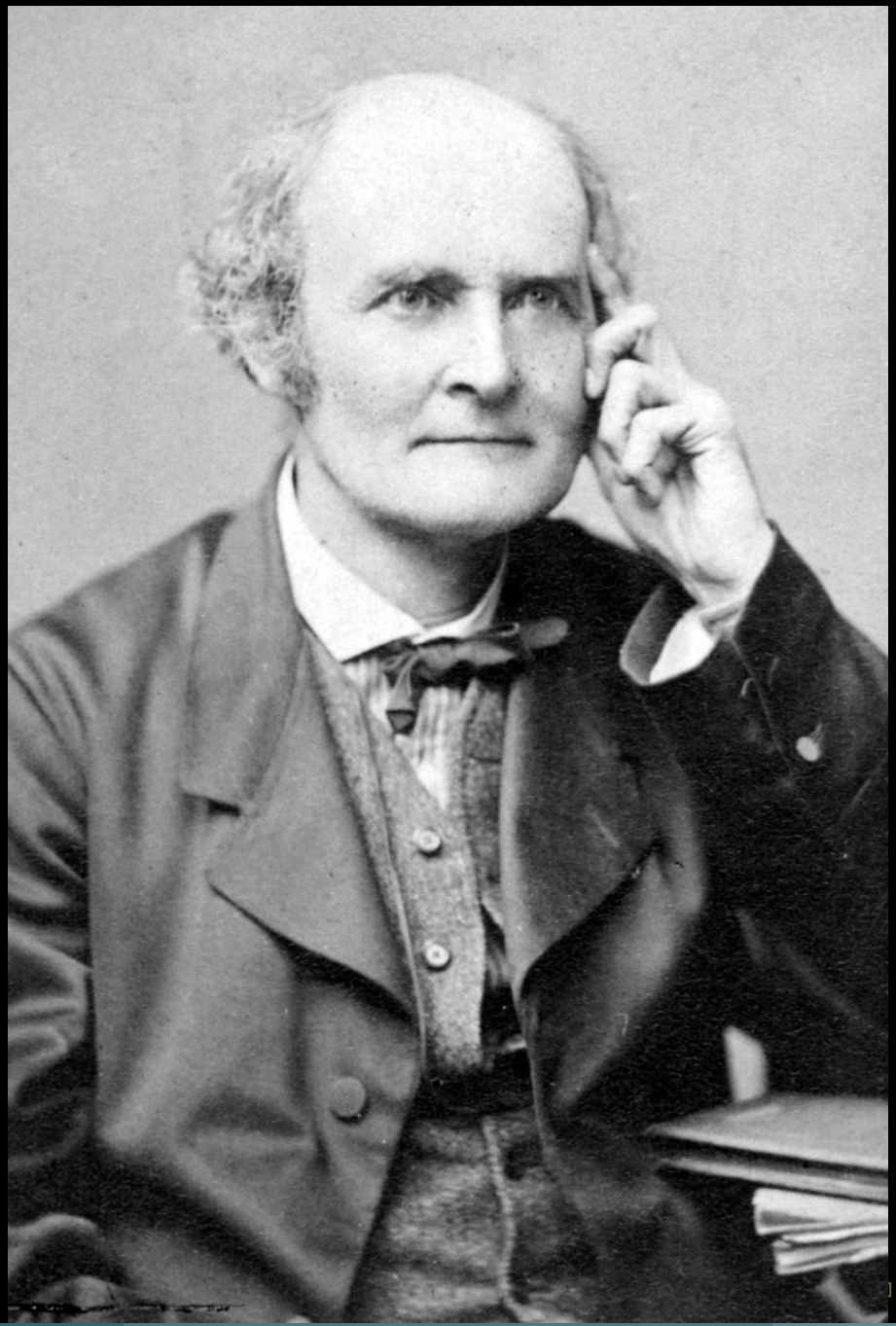
geb. am 01.03.1842 in Ellwangen
gest. am 17.04.1899 in Hannover

Professor der praktischen Geometrie und
höheren Geodäsie an der
Polytechnischen Schule in Karlsruhe
Professor der Geodäsie und praktischen
Geometrie an der TH Hannover

Arbeiten zur europäischen Gradmessung
Mitglied der Kaiserlichen Normal-Eichungskommision
Durchführung der Triangulation der Städte Hannover und Linden

1882 - 1899 Professor der Geodäsie und praktischen Geometrie an der TH Hannover





Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxplus)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxplus)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxdot)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxdot)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxdot)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxdot)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Nous appellerons *vecteur* une liste d'éléments d'un ensemble $\mathbb{A}$. Cet ensemble $\mathbb{A}$ est muni de deux opérations, $\boxplus$ et $\boxdot$, qui lui confèrent une structure d'**anneau** :

- $(\mathbb{A}, \boxplus)$ a une structure de groupe commutatif ;
- $(\mathbb{A}, \boxdot)$ a une structure de monoïde ;
- $\boxdot$ est distributive sur $\boxplus$.

On désigne souvent par $0_{\mathbb{A}}$ l'élément neutre de $\boxplus$ et par $1_{\mathbb{A}}$ l'élément neutre de $\boxdot$.
$(\mathbb{A}, \boxdot)$ n'étant pas forcément un groupe, tout le monde n'admet pas forcément un inverse par $\boxdot$.

Exercice 1

Quels sont les éléments inversibles de $(\mathbb{Z}, \cdot)$ *?*

```
class  (Eq a, Show a) => Num a  where
    (+), (-), (*)  :: a -> a -> a
    negate         :: a -> a
    abs, signum    :: a -> a
    fromInteger    :: Integer -> a
```

On notera $\mathbb{A}^p$ l'ensemble des vecteurs à coefficients dans $\mathbb{A}$ de *taille* p.

$$u \boxplus v = [a_1 \boxplus b_1, a_2 \boxplus b_2, ..., a_n \boxplus b_n]$$

$$ku = [k \boxdot a_1, k \boxdot a_2, ..., k \boxdot a_p]$$

$$u \boxdot v = [a_1 \boxdot b_1, a_2 \boxdot b_2, ..., a_p \boxdot b_p]$$

On notera $\mathbb{A}^p$ l'ensemble des vecteurs à coefficients dans $\mathbb{A}$ de *taille* p.

$$u \boxplus v = [a_1 \boxplus b_1, a_2 \boxplus b_2, ..., a_n \boxplus b_n]$$

$$ku = [k \boxdot a_1, k \boxdot a_2, ..., k \boxdot a_p]$$

$$u \boxdot v = [a_1 \boxdot b_1, a_2 \boxdot b_2, ..., a_p \boxdot b_p]$$

On notera $\mathbb{A}^p$ l'ensemble des vecteurs à coefficients dans $\mathbb{A}$ de *taille* p.

$$u \boxplus v = [a_1 \boxplus b_1, a_2 \boxplus b_2, ..., a_n \boxplus b_n]$$

$$ku = [k \boxdot a_1, k \boxdot a_2, ..., k \boxdot a_p]$$

$$u \boxdot v = [a_1 \boxdot b_1, a_2 \boxdot b_2, ..., a_p \boxdot b_p]$$

On notera $\mathbb{A}^p$ l'ensemble des vecteurs à coefficients dans $\mathbb{A}$ de *taille* p.

$$u \boxplus v = [a_1 \boxplus b_1, a_2 \boxplus b_2, ..., a_n \boxplus b_n]$$

$$ku = [k \boxdot a_1, k \boxdot a_2, ..., k \boxdot a_p]$$

$$u \boxdot v = [a_1 \boxdot b_1, a_2 \boxdot b_2, ..., a_p \boxdot b_p]$$

```
*Main> let u = [1, 2, 5, 4]
*Main> let v = [3, 4, 2, 3]
*Main> import Data.List
*Main Data.List> zipWith (*) u v
[3,8,10,12]
```

$\mathbb{A}^{n\times p}$: ensemble des matrices de n lignes et p colonnes à coefficients dans $\mathbb{A}$.

$$\begin{array}{c} \\ i \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & & j & & \\ \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,j} & \cdots & a_{1,p} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,j} & \cdots & a_{i,p} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,j} & \cdots & a_{n,p} \end{pmatrix} \end{array}$$

$\mathbb{A}^{n \times p}$: ensemble des matrices de n lignes et p colonnes à coefficients dans $\mathbb{A}$.

$$
\begin{array}{c} \\ i \end{array}
\begin{array}{c} \begin{array}{ccccc} & & & j & \end{array} \\
\begin{pmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,j} & \cdots & a_{1,p} \\
\vdots & \vdots & & \vdots & & \vdots \\
a_{i,1} & a_{i,2} & \cdots & a_{i,j} & \cdots & a_{i,p} \\
\vdots & \vdots & & \vdots & & \vdots \\
a_{n,1} & a_{n,2} & \cdots & a_{n,j} & \cdots & a_{n,p}
\end{pmatrix}
\end{array}
$$

```haskell
data Matrice a = Mat [[a]]
```