TD1 : Correction

Félix Martins-Ducasse

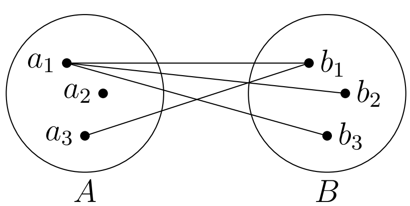
1 Relations et fonctions

Exercice 1.1 (*)

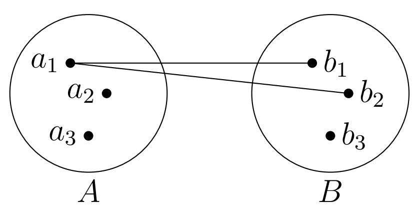
Énoncé

Donner un exemple de relation entre ensembles finis qui soit uniquement surjective ; uniquement injective ; uniquement entière ; uniquement déterministe. Donner un exemple de relation qui soit fonctionnelle mais pas co-fonctionnelle, puis co-fonctionnelle mais pas fonctionnelle.

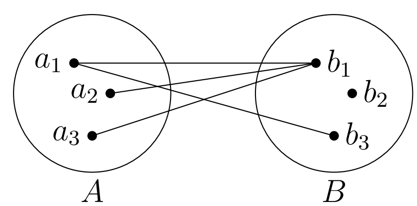
Correction



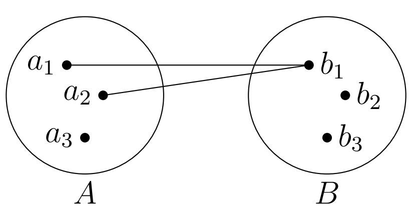
Uniquement Surjective



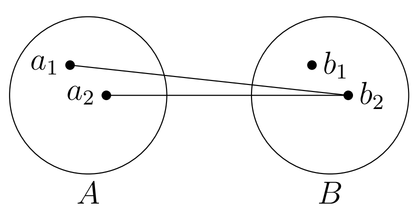
Uniquement Injective



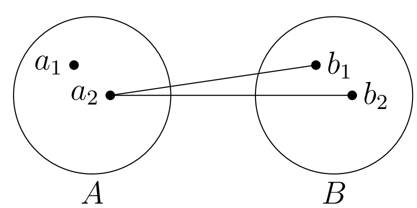
Uniquement Entière



Uniquement Déterministe



Fonctionnelle mais pas co-fonctionnelle



Co-fonctionnelle mais pas fonctionnelle

Exercice 1.2 (*)

Énoncé

Soient R et S des relations de même domaine et codomaine telles que R est entière et S déterministe. Démontrer que si $R \subseteq S$ alors $R = S$.

Correction

Il suffit de montrer que $S \subseteq R$ par antisymétrie, on a

(1) $Id \subseteq R; R^\circ$ Car R est entière
(2) $S^\circ; S \subseteq Id$ Car S est déterministe
Donc,

$$\begin{aligned} R \subseteq S &\implies S^\circ; R; R^\circ \subseteq S^\circ; S; R^\circ \\ &\implies S^\circ; Id^{(1)} \subseteq Id^{(2)}; R^\circ \\ &\iff S^\circ \subseteq R^\circ \\ &\iff S^{\circ\circ} \subseteq R^{\circ\circ} \\ &\iff S \subseteq R \quad \square \end{aligned}$$

Exercice 1.3 (*)

Énoncé

Soient $R : A \nrightarrow B$ et $S : A \nrightarrow C$ et $T : C \nrightarrow B$ des relations. L'extension $S \rhd R$ de R le long de S et le relèvement $R \lhd T$ de R le long de T satisfont

$$T \subseteq S \rhd R \iff S; T \subseteq R \iff S \subseteq R \lhd T$$

Utiliser ce résultat, et ce résulstat uniquement, pour démontrer les inclusions ci-dessous.

$$S; S \rhd R \subseteq R \qquad R \lhd T; T \subseteq R \qquad T \subseteq S \rhd (S; T) \qquad S \subseteq (S; T) \lhd T$$

Correction

$$\begin{aligned} S; S \rhd R \subseteq R &\iff S \rhd R \subseteq S \lhd R \\ R \lhd T; T \subseteq R &\iff R \lhd T \subseteq R \lhd T \\ T \subseteq S \rhd (S; T) &\iff S; T \subseteq S; T \\ S \subseteq (S; T) \lhd T &\iff S; T \subseteq S; T \end{aligned}$$

2 Constructions Ensemblistes

Exercice 2.1 (*)

Énoncé

Soient $R : A \nrightarrow B$ et $S_1, S_2 : B \nrightarrow C$ des relations. Donner un contre-exemple à l'inclusion $(R; S_1) \cap (R; S_2) \subseteq R; (S_1 \cap S_2)$

Correction

Montrons que l'autre sens de l'inclusion est vérifiée, soit

$$R : A \nrightarrow B \qquad S_1, S_2 : B \nrightarrow C$$

Preuve :

$$\begin{aligned} S_1 \cap S_2 \subseteq S_i \quad \forall i \in \{1,2\} &\implies R;(S_1 \cap S_2) \subseteq R;S_i \quad \forall i \in \{1,2\} \\ &\implies R;(S_1 \cap S_2) \subseteq (R;S_1) \cap (R;S_2) \quad \square \end{aligned}$$

Contre exemple pour l'autre sens de l'inclusion :
Soit $A = \{a\}$, $B = \{b_1, b_2\}$ et $C = \{c\}$, et les relations suivantes :

$$\begin{aligned} R &= \{(a, b_1), (a, b_2)\} \\ S_1 &= \{(b_1, c)\} \\ S_2 &= \{(b_2, c)\} \end{aligned}$$

Alors, on a

$$\begin{aligned} (R;S_1) \cap (R;S_2) &= \{(a, c)\} \\ R;(S_1 \cap S_2) &= \emptyset \end{aligned}$$

Donc,

$$(R;S_1) \cap (R;S_2) \nsubseteq R;(S_1 \cap S_2) \quad \square$$

2.1 Exercice 2.2 (*)

Énoncé

Écrire, dans un language fonctionnel de votre choix muni de types fonctions, produits et sommes, les biejctions canoniques de la figure ci dessous.

$$(A+B)+C \cong A+(B+C) \tag{1}$$
$$A+B \cong B+A \tag{2}$$
$$A+\mathbb{0} \cong A \tag{3}$$
$$(A\times B)\times C \cong A\times(B\times C) \tag{4}$$
$$A\times B \cong B\times A \tag{5}$$
$$A\times \mathbb{1} \cong A \tag{6}$$
$$A\times(B+C) \cong A\times B + A\times C \tag{7}$$
$$A\times \mathbb{0} \cong \mathbb{0} \tag{8}$$
$$C^{\mathbb{1}} \cong C \tag{9}$$
$$\mathbb{1}^C \cong \mathbb{1} \tag{10}$$
$$(A\times B)^C \cong A^C \times B^C \tag{11}$$
$$C^{\mathbb{0}} \cong \mathbb{1} \tag{12}$$
$$C^{A+B} \cong C^A \times C^B \tag{13}$$
$$\mathbb{0}^C \cong \mathbb{0} \text{ ssi } C \neq \mathbb{0} \tag{14}$$

```
module Empty = struct
  type t = |

  let exfalso : 'a. t -> 'a =
    fun x ->
    match x with
    | _ -> .
end

open Either

(* (A + B) + C = A + (B + C) *)

let bij1fwd : 'a 'b 'c.
              (('a, 'b) Either.t, 'c) Either.t ->
              ('a, ('b, 'c) Either.t) Either.t =
  function
  | Left (Left y) -> Left y
  | Left (Right y) -> Right (Left y)
  | Right y -> Right (Right y)

let bij1bwd : 'a 'b 'c.
              ('a, ('b, 'c) Either.t) Either.t ->
              (('a, 'b) Either.t, 'c) Either.t =
  function
  | Left x -> Left (Left x)
  | Right (Left x) -> Left (Right x)
  | Right (Right x) -> Right x

(* A + B = B + A *)

let bij2fwd : 'a 'b. ('a, 'b) Either.t -> ('b, 'a) Either.t =
  function
```

```
  | Left x -> Right x
  | Right x -> Left x

let bij2bwd : 'a 'b. ('b, 'a) Either.t -> ('a, 'b) Either.t =
  bij2fwd

(* A + 0 = A *)

let bij3fwd : 'a. ('a, Empty.t) Either.t -> 'a =
  function
  | Left x -> x
  | Right _ -> .

let bij3bwd : 'a. 'a -> ('a, Empty.t) Either.t =
  fun x -> Left x

(* (A * B) * C = A * (B * C) *)

let bij4fwd : 'a 'b 'c.
              ('a * 'b) * 'c
              ->
              'a * ('b * 'c)
  = fun ((x, y), z) -> (x, (y, z))

let bij4bwd : 'a 'b 'c.
              'a * ('b * 'c)
              ->
              ('a * 'b) * 'c
  = fun (x, (y, z)) -> ((x, y), z)

(* A * B = B * A *)

let bij5fwd : 'a 'b. ('a * 'b) -> ('b * 'a) =
  fun (x, y) -> (y, x)

let bij5bwd : 'a 'b. ('b * 'a) -> ('a * 'b) =
  bij5fwd

(* A * 1 = A *)

let bij6fwd : 'a. 'a * unit -> 'a =
  fun (x, ()) -> x

let bij6bwd : 'a. 'a -> 'a * unit =
  fun x -> (x, ())

(* A * (B + C) = A * B + A * C *)

let bij7fwd : 'a 'b.
              'a * ('b, 'c) Either.t ->
              ('a * 'b, 'a * 'c) Either.t =
  function
  | (x, Left y) -> Left (x, y)
  | (x, Right y) -> Right (x, y)

let bij7bwd : 'a 'b.
              ('a * 'b, 'a * 'c) Either.t ->
```

```
                'a * ('b, 'c) Either.t =
  function
  | Left (x, y) -> (x, Left y)
  | Right (x, y) -> (x, Right y)

(* A * 0 = 0 *)

let bij8fwd : 'a. ('a * Empty.t) -> Empty.t =
  (* fun (_, x) -> x *)
  function
  | _ -> .

let bij8bwd : 'a. Empty.t -> ('a * Empty.t) =
  function
  | _ -> .

(* C^1 = C *)

let bij9fwd : 'c. (unit -> 'c) -> 'c =
  fun f -> f ()

let bij9bwd : 'c. 'c -> (unit -> 'c) =
  fun x -> fun () -> x

(* 1^C = 1 *)

let bij10fwd : 'c. ('c -> unit) -> unit =
  fun _ -> ()

let bij10bwd : 'c. unit -> ('c -> unit) =
  fun () -> fun _ -> ()

(* (A * B)^C = A^C * B^C *)

let bij11fwd : 'a 'b 'c.
               ('c -> ('a * 'b))
               ->
               ('c -> 'a) * ('c -> 'b)
  = fun f -> ((fun x -> fst (f x)), (fun x -> snd (f x)))

let bij11bwd : 'a 'b 'c.
               ('c -> 'a) * ('c -> 'b)
               ->
               ('c -> ('a * 'b))
  = fun (f, g) -> fun x -> (f x, g x)

(* C^0 = 1 *)

let bij12fwd : 'c.
               (Empty.t -> 'c)
               ->
               unit
  = fun _ -> ()

let bij12fwd : 'c.
               unit
               ->
```

```
                (Empty.t -> 'c)
  = fun () -> (function _ -> .)

(* C^(A + B) = C^A * C^B *)

let bij13fwd : 'a 'b 'c.
                (('a, 'b) Either.t -> 'c)
                ->
                ('a -> 'c) * ('b -> 'c)
  = fun f -> ((fun x -> f (Left x)),
               (fun x -> f (Right x)))

let bij13bwd : 'a 'b 'c.
                ('a -> 'c) * ('b -> 'c)
                ->
                (('a, 'b) Either.t -> 'c)
  = fun (f, g) -> function Left x -> f x
                          | Right x -> g x

(* 0^C = 0     si C n'est pas vide  *)

let bij14fwd : 'c. 'c ->
                ('c -> Empty.t)
                ->
                Empty.t
  = fun x f -> f x

let bij14fwd : 'c. 'c ->
                  Empty.t
                ->
                ('c -> Empty.t)
  = fun x y -> match y with _ -> .
```