Examen. Session 2. Documents autorisés. Durée 1h30.

1 Structure de données

On s'intéresse aux ensembles d'entiers naturels, représentés comme des arbres lexicographiques binaires où les branches encodent la décomposition en base 2 des entiers présents dans l'ensemble, bit de poids fort en premier.

Ainsi, l'ensemble $\{0,1,3,5,8\}$, vu comme l'ensemble de mots binaires $\{\text{""}_2,\text{"1"}_2,\text{"11"}_2,\text{"101"}_2,\text{"1000"}_2\}$, sera représenté par l'arbre de la figure 1 :

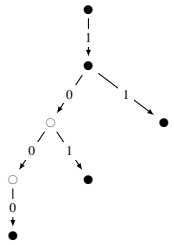


FIGURE 1 – Un exemple d'arbre

On note que le bit de poids fort est toujours '1', sauf pour l'entier 0 représenté canoniquement par le mot binaire vide. Dans chaque noeud est indiqué si la branche courante est bien un mot binaire de l'ensemble (●) ou bien un préfixe strict d'un tel mot (○).

Le type de données suivant ensnat permet de manipuler de tels arbres. L'ensemble $\{0,1,3,5,8\}$ est représenté par la variable exemple.

```
type ensnat = Node of bool * ensnat option * ensnat option
let node b l r = Some (Node (b, l, r))
let leaf = node true None None
let exemple =
  node true
       None
       (node true
             (node false
                   (node false
                         leaf
                         None)
                   leaf )
             leaf )
```

Exercice 1 (Fonctions élémentaires)

1. Écrire la fonction cardinal : ensnat -> int qui calcule le cardinal d'un ensemble d'entiers.
2. Écrire la fonction binaire : int -> bool list qui calcule la décomposition binaire d'un entier naturel. Le booléen true représente le bit '1'.

Exercice 2 (appartenance & ajout)

1. Écrire la fonction appartient : int -> ensnat -> bool, qui teste l'appartenance d'un entier naturel à un ensemble.
2. Écrire la fonction ajout : int -> ensnat -> ensnat, ajoutant un entier naturel à un ensemble.

Exercice 3 (retrait & normalisation)

1. Écrire la fonction retrait : int -> ensnat -> ensnat qui retire un entier naturel d'un ensemble. On pourra laisser des branches inutiles dans l'arbre.
2. Le cas échéant, améliorer retrait afin qu'elle produise un arbre sans branches inutiles, i.e. normalisé.

2 Itérateurs monadique

On donne la définition de type suivante pour des itérateurs fonctionnels ordinaires :

```
(* le type des itérateurs produisant des éléments de type 'a *)
(* la fonction produit le prochain élément s'il existe, ainsi que l'état suivant de l'itérateur *)
type 'a iter = Next of unit -> ('a * 'a iter) option
```

On souhaite munir le type `'a iter` des opérations formant une monade additive.

Exercice 4 (Monade) Définir les opérations formant une monade :

1. `map : ('a -> 'b) -> 'a iter -> 'b iter`
2. `return : 'a -> 'a iter`
3. `bind : 'a iter -> ('a -> 'b iter) -> 'b iter`

Exercice 5 (Structure additive) Définir les opérations additives :

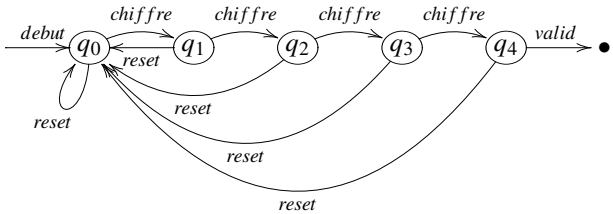
1. `zero : 'a iter`
2. `plus : 'a iter -> 'a iter -> 'a iter`

3 Typage avançé

On cherche à décrire le comportement d'un automate bancaire lisant un code PIN à 4 chiffres. On se donne l'interface basique suivante, représentant les différentes opérations permises. Le type `t` est l'état interne de l'automate. `debut` initialise l'automate (insertion de la carte bancaire), `chiffre` ajoute un nouveau chiffre entré au clavier, `reset` réinitialise l'automate et `valid` teste si le code entré est correct, seulement si on a entré 4 chiffres à la suite.

```
module type PIN =
sig
type t
  val debut   : unit -> t
  val chiffre : t -> t
  val reset   : t -> t
  val valid   : t -> bool
end
```

Les transitions "légales" sont données dans le schéma suivant :



On souhaite restreindre les opérations afin de ne permettre que des transitions "légales". On ajoute au type `t` un paramètre représentant l'état interne de l'automate, soit **type** `_ t`.

Exercice 6 (Modélisation du comportement)

1. Comment représenter les états internes $q_0, \ldots, q_4$ par des types ?
2. Modifier la signature des opérations de `PIN` pour n'autoriser que les transitions légales.

On modifie le comportement afin de permettre l'entrée d'autant de chiffres que voulu, la validation ne pouvant se faire que lorsqu'au moins 4 chiffres on été entrés.

Exercice 7 (Deuxième comportement) Modifier la signature des opérations de PIN pour modéliser ce nouveau comportement.

On souhaite mémoriser dans le type t les quatre derniers chiffres entrés. On se donne les nouvelles déclarations de types suivantes :

```
type ('a, 'b, 'c, 'd) t
type zero
type un
⋮
type neuf
type _ chfr = Zero : zero chfr | Un : un chfr | ... | Neuf : neuf chfr
```

Les paramètres 'a, ..., 'd du type t sont les chiffres entrés. À l'état initial, on se comportera comme si on avait déjà entré 4 fois le chiffre '0'. On peut donc valider depuis l'état initial (et tout autre état).

Exercice 8 (Troisième comportement) Modifier la signature des opérations pour modéliser ce nouveau comportement, en ajoutant à la signature de de chiffre un argument de type ... chfr, représentant le chiffre entré.