PFITA – Cours/TD 9

UPS – L3 Info – Semestre 5

1 Types abstraits de données

On considère les 2 TAD suivants :

```
spec ELEM
sortes Elem
operations
  a, b: Elem

spec ENS etend ELEM
sortes Ens
operations
  {_} : Elem -> Ens
  _ \/ _ : Ens * Ens -> Ens
  choix: Ens -> Elem
variables
  e1,e2: Ens
  x: Elem
axiomes
  e1 \/ e2 = e2 \/ e1
  choix({x}) = x
  choix(e1 \/ e2) = choix(e1)
```

Montrer que ENS n'est pas hiérarchiquement consistant par rapport à ELEM.

```
À partir des axiomes de ENS, on peut déduire que a = b,
alors que cette propriété n'est pas démontrable dans ELEM :

a = choix({a}) = choix({a} \/ {b})
               = choix({b} \/ {a}) = choix({b}) = b
```

(1)

2 Signatures et modules OCaml

Compléter la spécification suivante des listes génériques avec curseur :

```
module type tCLISTE = sig
type 'a t
val vide: 'a t
val estADroite: 'a t -> bool
val estAGauche: 'a t -> bool
val droite: 'a t -> 'a t
val gauche: 'a t -> 'a t
val courantADroite: 'a t -> 'a
val insererADroite: 'a -> 'a t -> 'a t
val supprimerADroite : 'a t -> 'a t
val courantAGauche: 'a t -> 'a
val insererAGauche: 'a -> 'a t -> 'a t
val supprimerAGauche : 'a t -> 'a t
end
```

```
(* AXIOMES *) estADroite(vide) = true ; estAGauche(vide) = true
estADroite(insererADroite(e,l)) = false
estAGauche(insererADroite(e,l)) = estAGauche(l)
courantADroite(insererADroite(e,l)) = e
courantAGauche(insererADroite(e,l)) = courantAGauche(l)
supprimerADroite(insererADroite(e,l)) = l
supprimerAGauche(insererADroite(e,l)) = insererADroite(e,supprimerAGauche(l))
droite(insererADroite(e,l)) = insererAGauche(e,l)
gauche(insererADroite(e,l)) = let (ll,g0)=(supprimerAGauche l,courantAGauche l) in
  insererADroite(g0,insererADroite(e,ll))
(* et symétriquement *)
estAGauche(insererAGauche(e,l)) = false
estADroite(insererADroite(e,l)) = estADroite(l)
courantAGauche(insererAGauche(e,l)) = e
courantADroite(insererAGauche(e,l)) = courantADroite(l)
supprimerAGauche(insererAGauche(e,l)) = l
supprimerADroite(insererAGauche(e,l)) = insererAGauche(e,supprimerADroite(l))
gauche(insererAGauche(e,l)) = insererADroite(e,l)
droite(insererAGauche(e,l)) = let (d0,ll)=(supprimerADroite l,courantADroite l) in
  insererAGauche(d0,insererAGauche(e,ll) (* ET LES PRÉCONDITIONS... *)
```

(2)

Compléter l'implantation suivante, en utilisant des couples de pile :

```
  module CLISTE (P : tPILE): tCLISTE = struct
type 'a t = 'a P.pile * 'a P.pile
let vide = (P.vide, P.vide)
let estADroite (_,d) = P.estVide d
let estAGauche (g,_) = P.estVide g
let droite (g,d) =
  if P.estVide d then failwith "A␣droite"
  else (P.empiler (P.sommet d) g, P.depiler d)
let gauche (g,d) =
  if P.estVide g then failwith "A␣gauche"
  else (P.depiler g, P.empiler (P.sommet g) d)

let courantADroite (g,d) =
  if P.estVide d then failwith "A␣droite"
  else P.sommet d
let insererADroite e (g,d) = (g, P.empiler e d)
let supprimerADroite (g,d) =
  if P.estVide d then failwith "A␣droite"
  else (g, P.depiler d)
(* et similairement pour courantAGauche, insererAGauche, supprimerAGauche : *)
let courantAGauche (g, d) =
  if P.estVide g then failwith "A␣gauche"
  else P.sommet g
let insererAGauche e (g,d) = (P.empiler e g, d)
let supprimerAGauche (g, d)
  if P.estVide g then failwith "A␣gauche"
  else (P.depiler g, d)
end
```

(3)