TRAVAUX PRATIQUES : série n°3

Implémenter les types abstraits **SET** et **BAG**

Objectif

L'objectif de ce TP est de fournir une implémentation **prouvée** (par application d'une **Vérification Formelle**) pour les types de structures ensemblistes très utilisées en ingénierie du logiciel à savoir, les **Set** et les **Bag**.

La méthode qui sera appliquée s'appuie sur les spécifications Casl proposées en cours.

La démarche doit **obligatoirement** dérouler en suivant les **5** dernières phases du cycle de développement initié lors du TP n°1 sur le type abstrait des polynômes, à savoir :

Phase 2 : **Spécification Casl du type abstrait (elle est fournie et doit être éditée sous emacs)**
Phase 3 : **Validation de la spécification sous Hets Casl**.
Phase 4 : **Spécification des opérations du type**
Phase 5 : **Implémentation des opérations du type**
Phase 6 : **Validation de l'implémentation**

Dans la plupart des applications, les objets abstraits de type Set ou Bag sont des objets dynamiques (très variables) Aussi, leur implémentation à l'aide d'objets concrets de type **listes chaînées** est plus efficace que celle plus classique qui utilise des tableaux.

I- Type abstrait SET

a)Commencer par implémenter la spécification **canonique** du type SET

```
%% signature
spec SET0 [sort Elem] =
      generated type Set[Elem] ::= setVide
                                  |ajouter(Set[Elem]; Elem)
     pred
     appartient: Elem × Set[Elem]

%% sémantique
    ∀ x, y: Elem; M, N: Set[Elem]
     • ¬ appartient(x, setVide)
     • appartient(x, ajouter(M,y)) ⇔ x = y  ∨  appartient(x, M)
     • M = N ⇔ ∀ x: Elem • appartient(x ,M) ⇔ appartient(x,N)
end
```

b) implémenter, ensuite, la spécification plus complète obtenue par **extension** de la spécification canonique précédente.

```
spec SET [sort Elem] given NAT=
      SET0 [sort Elem]
      then
      pred
            estVide: Set[Elem];
      op
            enlever : Set[Elem] × Elem ⟶ Set[Elem]

∀ x, y: Elem; M: Set[Elem]
      • estVide(M) ⇔ M = setVide
%% le constructeur enlever est défini par induction à partir de setVide et ajouter
      • enlever(setVide, y) = setVide
      • enlever(ajouter(M,x), y)) = M when x = y   else ajouter(enlever(M,y),x)
end
```

c)Spécification des opérations du type

setVide() r: Set[Elem]
pré: true
post: ∀ x: Elem
- **estVide(r)**
- ¬ **appartient**(x, **r**)
- **enlever**(**r**, x) = **r**

ajouter(M:Set[Elem], x: Elem) **r**: Set[Elem]
pré:
post: ∀y: Elem
- **appartient**(y,**r**) ⇔ x = y ∨ **appartient**(y, M)

enlever(M:Set[Elem] , x:Elem) **r**:Set[Elem]
pré: true
post: ∀ y
- M = **setVide()** ⇒ **r** = **setVide()**
- x = y ⇒ **enlever**(M,y)= **r**

appartient(x:Elem, M: Set[Elem]) **r**: booléen
pré: true
post
∀ y: Elem; M0,N: Set[Elem]
- **estVide**(M) ⇒ ¬ **r**
- M= **ajouter**(M0,y) ⇒ **r** ⇔ x = y ∨ **appartient**(x, M0)
- M **=** N ⇒ **appartient**(x,N) ⇔ **r**

estVide(M:Set[Elem]) **r**:Booléen
pré: true
post: **r** ⇔ M = **setVide()**

II- Type BAG

a)Commencer par implémenter la spécification **canonique** suivante.

```
spec BAG0 [sort Elem] given NAT =
    generated type Bag[Elem] ::= bagVide |
                                 ajouter(Bag[Elem]; Elem)
     op
      freqence : Bag[Elem] × Elem ⟶ Nat

     ∀ x,y: Elem; M, N:Bag[Elem]
     • freqence(bagVide, y) = 0
     • freqence(ajouter(M,x), y) = freqence(M,y)+1 when x = y  else freqence(M, y)
      • M = N ⇔ ∀ x: Elem •  freqence(M,x) = freqence(N,x )

end
```

b) Implémenter ensuite la spécification suivante obtenue par **extension** de la spécification canonique précédente.

```
spec BAG [sort Elem] given Nat =
        BAG0 [sort Elem]
then
        pred
        estVide: Bag[Elem]
        op
        enlever: Bag[Elem] × Elem ⟶ Bag[Elem]

        ∀ x,y:Elem; M,N:Bag[Elem]
        • estVide(M) ⇔ M =bagVide
        • N = enlever(M,x) ⇔ ∀ y: Elem •(freqence(N,y) = freqence(M,x) −1 when x = y
                                                                    else freqence(M,y)
end
```

c)Spécification des opérations du type

bagVide() **r**: Bag[Elem]
pré: true
post:
∀ y: Elem
- **freqence**(**r**, y) = 0
- **estVide**(**r**)

ajouter(M:Bag[Elem], x:Elem) **r**: Bag[Elem]
pré: true
post: ∀ y: Elem
- ¬**estVide**(**r**)
- x = y ⇒ **freqence**(**r**, y) = **freqence**(M,y)+1
- x ≠ y ⇒ **freqence**(**r**, y) = **freqence**(M,y)

enlever(M:Bag[Elem], x: Elem) **r**: Bag[Elem]
pré: true
post: ∀ y:Elem

- **freqence**(**r**, y) = 0 ⇔ **estVide**(**r**)
- x = y ⇒ **freqence**(**r**, y) = **freqence**(M,y)-1
- x ≠ y ⇒ **freqence**(**r**, y) = **freqence**(M,y)

freqence(M: Bag[Elem], x: Elem) **r**:Nat
pré: true
post: ∀ y: Elem; N:Bag[Elem]

- M= **bagVide()** ⇒ **r** = 0
- x = y ⇒ M= **ajouter**(N,y) ⇒ **r** = **freqence**(N,y)+1
- x ≠ y ⇒ M= **ajouter**(N,y) ⇒ **r** = **freqence**(N,x)
- x = y ⇒ M= **enlever**(N,y) ⇒ **r** = **freqence**(N,y)-1
- x ≠ y ⇒ M= **enlever** (N,y) ⇒ **r** = **freqence**(N,x)
- M = N ⇔ **r** = **freqence**(N,x)

estVide(M:Bag[Elem]) **r**:Booléen
pré: true
post: **r** ⇔ M =**bagVide()**