Les types abstraits de données

Déclaration des principaux TAD

Nathanaël Cottin

Avant-propos

- Une spécification algébrique décrit formellement un type de données ainsi que son comportement sous forme d'une liste d'opérations et d'axiomes (propriétés)

Définition des TAD (1/2)

- Les types abstraits de données permettent de représenter des abstractions courantes
- Ils ne sont pas liés à une quelconque mise en œuvre dans un langage de programmation
- Ils se composent :
  - D'opérations (de base et évoluées)
  - De propriétés (comportement des opérations)

Définition des TAD (2/2)

- Les opérations ne peuvent être employées que lorsque leurs préconditions sont remplies
- Les propriétés sont exprimées en combinant les opérations : la spécification est doit être cohérente (impossibilité de démontrer par récurrence sur les axiomes que VRAI = FAUX)

Le TAD « liste »

Présentation

- Représentation d'une suite ordonnée ou non d'éléments
- Déclaration du type « liste » composé de données unitaires de type « element »
- Instanciation :

$$\begin{cases} \forall l \in liste \\ \forall e \in element \end{cases}$$

Opérations de base

- créer : ø → liste
- estVide : liste → booléen
- tête : liste → element
- queue : liste → liste
- ajouter : element x liste → liste

Propriétés des opérations de base

- estVide(créer()) = VRAI
- estVide(ajouter(e, l)) = FAUX
- tête(ajouter(e, l)) = e
- queue(ajouter(e, l)) = l
- ajouter(tête(l), queue(l)) = l

Préconditions

- tête(l) → ¬estVide(l)
- queue(l) → ¬estVide(l)

Les opérations « tête » et « queue » ne peuvent être appelées que si la liste est non vide

Opérations évoluées (exemples)

- taille : liste → entier
- appartient : element x liste → booléen
- dupliquer : liste → liste
- vider : liste → liste
- inverser : liste → liste
- insérer : element x entier x liste → liste
- supprimer : entier x liste → liste

Propriétés des opérations évoluées

- taille(créer()) = 0
- taille(l) = taille(queue(l)) + 1

- inverser (créer()) = créer()
- inverser (l) =
  ajouterQueue(tête(l), inverser(queue(l)))

Longueur d'une liste : algorithme

Version itérative :

```
fonction taille : liste l → entier
    longueur = 0 : entier
début
    tant que ¬estVide(l) faire
        longueur ← longueur + 1
        l ← queue(l)
    fait
    taille ← longueur
fin
```

Version récursive :

```
fonction taille : liste l → entier
    t : entier
début
    t ← si estVide(l) alors
        0
    sinon
        taille( queue(l) ) + 1
    taille ← t
fin
```

Inversion : algorithme

```
fonction inverser : liste l → liste
    l2 = créer() : liste
début
    tant que ¬estVide(l) faire
        l2 ← ajouter( tête(l), l2 )
        l  ← queue(l)
    fait
    inverser ← l2
fin
```


Le TAD « file »

Présentation

- Représentation d'une file (nécessairement ordonnée) d'éléments
- Déclaration du type « file » composé de données unitaires de type « element »
- La file peut être FIFO ou LIFO
- Instanciation :

$$\begin{cases} \forall f \in file \\ \forall e \in element \end{cases}$$

Opérations de base

- créer : ø → file
- estVide : file → booléen
- courant : file → element
- ajouter : element x file → file
- consommer : file → file

Propriétés des opérations de base

- estVide(créer()) = VRAI
- estVide(ajouter(e, f)) = FAUX
- Si LIFO :

  courant(ajouter(e, f)) = e
- Si FIFO :

  courant(ajouter(e, créer())) = e

  courant(ajouter(e, f))) = courant(f)
- consommer(ajouter(e, f)) = f

Préconditions

- courant(f) → ¬estVide(f)
- consommer() → ¬estVide(f)

Les opérations « courant » et « consommer » ne peuvent être appelées que si la file est non vide

Opérations évoluées (exemples)

- taille : file → entier
- appartient : element x file → booléen
- dupliquer : file → file
- vider : file → file
- inverser : file → file

Propriétés des opérations évoluées

- taille(créer()) = 0
- taille(f) = taille(consommer(f)) + 1
- taille(vider(f)) = 0
- appartient(e, créer()) = FAUX
- appartient(e, ajouter(e, f)) = VRAI
- dupliquer(créer()) = créer()
- courant(dupliquer(ajouter(e, f))) = e