UFR IM²AG

UNIVERSITÉ Grenoble Alpes

L3 Informatique
Année 2017–2018

UE ALGO5 — TD2 — Séance 8 : Arbres n-aires

On s'intéresse ici à l'implémentation d'un type abstrait «Arbre n-aire», en utilisant le type «Arbre binaire» vu lors de la séance précédente.

Un arbre n-aire est ici implémenté comme suit :
- le «fils aîné» d'un nœud n-aire est implémenté par le «fils gauche» du nœud binaire ;
- le «frère cadet» d'un nœud n-aire est implémenté par le «fils droit» du nœud binaire.

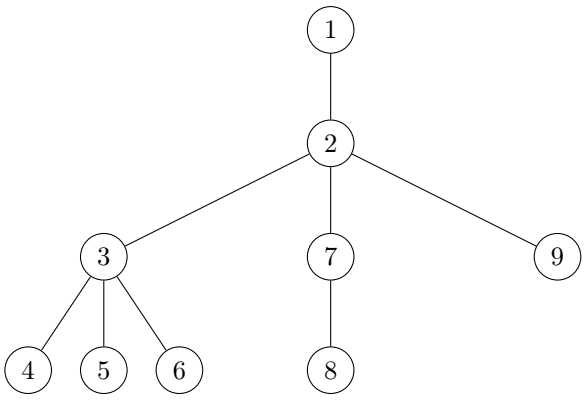
Introduction

Il existe différentes manières d'implémenter un arbre n-aire. Le principe repose sur le fait qu'un arbre est un couple (élément racine, liste d'arbres). Cette liste d'arbres peut être implémentée comme n'importe quelle liste : représentation contiguë dans un tableau, liste chaînée, etc. Dans la suite du TD on choisit de représenter une liste par un arbre.

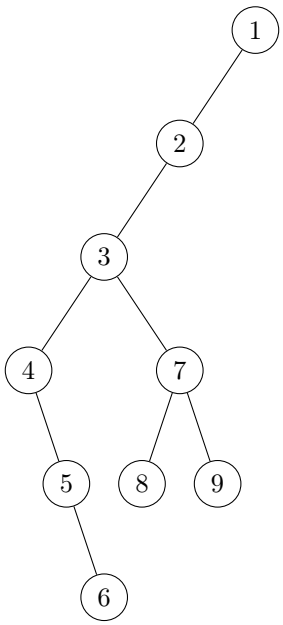
Il s'agit ici de montrer qu'une même structure (un arbre binaire) peut implémenter en même temps plusieurs types abstraits : ici, des arbres n-aires et des listes d'arbres, les deux types étant implémentés par un arbre binaire. L'exercice consiste à ne pas construire une liste d'arbres, mais d'utiliser la structure sous-jacente, avec des primitives de manipulation différentes selon le type abstrait.

Exercice 1.

Q 1. Traduisez l'arbre ci-dessous dans sa représentation binaire :



Corrigé —



Exercice 2.

Q 2. Implémentez les primitives du type abstrait «arbre n-aire», telles que définies ci-dessous.

```
Élément : un type
Arbre : un type
ListeArbre : un type

ArbreVide
{ Données : aucun
  Résultat : un Arbre vide}

NouveauNœud
{ Données : un Élément x, une ListeArbre L
  Résultat : un Arbre constitué du nœud x, dont les fils sont les éléments de L
  Effet de bord : un nouveau nœud a été créé }

EstArbreVide
{ Données : un Arbre A
  Résultat : un booléen vrai ssi A est un Arbre vide }

Elem
{ Données : un Arbre A
  Résultat : l'Élément associé àla racine de A
  Pré-condition : A est non vide }


```

```
   ListeFils
{ Données : un Arbre A
   Résultat : une ListeArbre
   description : A doit être non vide, renvoie la liste des fils associée àla racine de A }

{ Manipulation des listes d'arbres }

ListeVide
   { Données : aucun
   Résultat : une ListeArbre vide }

Cons
   { Données : un Arbre A, une ListeArbre L
   Résultat : une ListeArbre constituée de l'arbre A, suivi de la liste L. }

EstListeVide
   { Données : une ListeArbre L
   Résultat : un booléen vrai ssi L est vide. }

Premier
   { Données : une ListeArbre L
   Résultat : un Arbre, renvoie le premier arbre de la liste L
   Pré-condition : L non vide }

   Suivants
{ Données : une ListeArbre L
   Résultat : une ListeArbre, renvoie la liste des arbres suivants le premier.
   Pré-condition : L non vide }
```

Corrigé —

Implémentation : le plus difficile à comprendre est Premier(L), qui retourne L : on peut s'aider d'un schéma montrant que l'arbre binaire représentant la liste d'arbres L et son premier élément sont bien les mêmes ! Explication : le type change, et par conséquent l'ensemble des primitives applicables sur L. Si L est une liste d'arbres, on peut appliquer la primitive Suivant et récupérer le fils droit de l'arbre binaire sous-jacent. Si par contre L est un arbre, aucune primitive ne permet de partir sur ce fils droit...

```
Arbre : le type ArbreBin
ListeArbre : le type ArbreBin

ArbreVide:
    retourner ArbreBinVide

NouveauNoeud(x,L):
    retourner NouveauNoeudBin(L,x,ArbreBinVide)

EstArbreVide(A):
    retourner EstArbreBinVide(A)

Elem(A):
    retourner ElemBin(A)

ListeFils(A):
    retourner FGauche(A)

ListeVide:
    retourner ArbreBinVide

Cons(A,L):
    retourner NouveauNoeudBin(FGauche(A),Elem(A),L)
```

```
25 EstListeVide(L):
       retourner EstArbreBinVide(L)
27
   Premier(L):
29     retourner L

31 Suivants(L):
       retourner FDroit(L)
```
