Méthodes de conception de programmes
Devoir 2 : 1, 2, 3... Arbres !

Alexandre Gobeaux[a], Louis Navarre[a], Gilles Peiffer[a]

[a] *École Polytechnique, Université catholique de Louvain, Place de l'Université 1, 1348 Ottignies-Louvain-la-Neuve, Belgique*

Abstract

Ce papier donne les invariants de représentation, la fonction d'abstraction et les spécifications des fonctions `insert` et `join` pour une implémentation des arbres 2-3 basée sur Sedgewick and Wayne (2011) et Wikipedia contributors (2018).

1. Invariant de représentation

Commençons par définir quelques fonctions auxiliaires :
- size(T) : donne le nombre de nœuds d'un arbre T ;
- height(T) : donne la hauteur d'un arbre T ;
- type(T) : donne le nombre de sours-arbres du nœud source de l'arbre T.

Afin d'alléger la notation de l'invariant de représentation $\texttt{ok}(T)$, voici quelques fonctions supplémentaires.
- Si le nœud source de T est un 2-nœud, alors L et R dénotent respectivement le sous-arbre de gauche et de droite de T, alors que a dénote la donnée de son nœud source.
- Si le nœud source de T est un 3-nœud, alors L, M et R dénotent respectivement le sous-arbre de gauche, du milieu et de droite de T, alors que $a < b$ sont les données du nœud source.

$$f(T) = \Big(\text{size}(L) > 0 \wedge \text{size}(R) > 0\Big) \wedge \Big(\text{height}(L) = \text{height}(R)\Big) \wedge \Big(\forall \lambda \in L, \varrho \in R : \lambda < a \leq \varrho\Big), \tag{1}$$

$$g(T) = \Big(\text{size}(L) > 0 \wedge \text{size}(M) > 0 \wedge \text{size}(R) > 0\Big) \wedge \Big(\text{height}(L) = \text{height}(M) = \text{height}(R)\Big) \wedge \Big(\forall \lambda \in L, \mu \in M, \varrho \in R : \lambda < a \leq \mu < b \leq \varrho\Big). \tag{2}$$

L'invariant de représentation est alors donné par

$$\texttt{ok}(T) \equiv \text{size}(T) = 0 \vee \Big(\text{type}(T) = 2 \wedge f(T)\Big) \vee \Big(\text{type}(T) = 3 \wedge g(T)\Big). \tag{3}$$

2. Fonction d'abstraction

La fonction d'abstraction `abs`() est donnée par

$$\texttt{abs}() = \ . \tag{4}$$

3. Spécifications formelles

3.1. Spécification de `insert(t: Tree, i: int) returns (t': Tree)`

3.1.1. Précondition

La précondition est simplement que l'invariant de représentation soit respecté.

```
@Pre ≡ ok(t)
```

3.1.2. Postcondition

Comme la fonction `insert` doit conserver le rep-invariant, il fait également partie de la postcondition. En plus de cela, on requiert également que le multiensemble des éléments du nouvel arbre soit égal au multiensemble résultant de l'union de l'arbre original et de `i`, où les multiensemble sont définis comme dans Blizard (1991).

```
@Post ≡ ok(t') ∧ elements(t') = elements(t) ∪ multiset(i)
```

Ici, `elements` est une fonction qui prend en argument un arbre 2-3 et retourne le multiensemble de ses éléments.

Email addresses: alexandre.gobeaux@student.uclouvain.be (Alexandre Gobeaux), navarre.louis@student.uclouvain.be (Louis Navarre), gilles.peiffer@student.uclouvain.be (Gilles Peiffer)

3.2. Spécification de `join(t1: Tree, t2: Tree) returns (t: Tree)`

3.2.1. Précondition

La fonction `join` a comme précondition que les deux arbres pris en argument soient des arbres 2-3 valables.

```
@Pre ≡ ok(t1) ∧ ok(t2)
```

3.2.2. Postcondition

Le résultat de la fonction `join` doit également être un arbre 2-3 valable. En plus de cela, on requiert également que le multiensemble des éléments du nouvel arbre soit égal au multiensemble résultant de l'union des deux arbres initiaux.

```
@Post ≡ ok(t) ∧ elements(t) = elements(t1) ∪ elements(t2)
```

La fonction `elements` est la même que pour `insert`.

Références

Blizard, W. D., 1991. The development of multiset theory. Mod. Log. 1 (4), 319–352.
URL https://projecteuclid.org:443/euclid.rml/1204834739

Sedgewick, R., Wayne, K., 2011. Algorithms, 4th Edition. Addison-Wesley, 21415 Network Place, Chicago, IL 60673, United States.

Wikipedia contributors, 2018. 2–3 tree — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=2%E2%80%933_tree&oldid=857850249, [Online ; accessed 20-April-2019].