Programmation Fonctionnelle

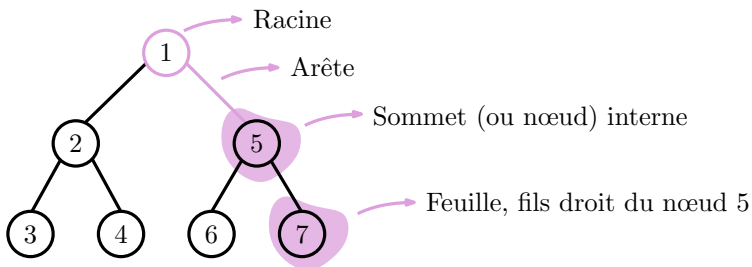
Séance 5 : Arbres

Alexandros Singh
Université Paris 8

9 novembre 2023

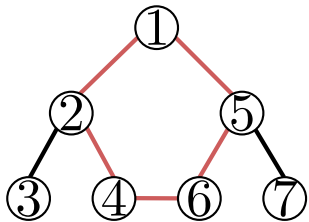
Definition (Arbre binaire)

Un arbre binaire (enraciné) est une collection de sommets (étiquetés) et d'arêtes qui les relient (deux à la fois), sans aucun cycle, tel que chaque sommet est soit : interne (a deux fils) soit une feuille (n'a pas de fils). Il est muni d'un sommet marqué, la racine.



Definition (Arbre binaire)

Un arbre binaire (enraciné) est une collection de sommets (étiquetés) et d'arêtes qui les relient (deux à la fois), sans aucun cycle, tel que chaque sommet est soit : interne (a deux fils) soit une feuille (n'a pas de fils). Il est muni d'un sommet marqué, la racine.



Il existe de nombreuses façons de représenter un arbre binaire en tant que structure de données. Beaucoup d'entre elles utilisent la définition alternative suivante (notez la récursivité !) :

Definition (Arbre binaire)

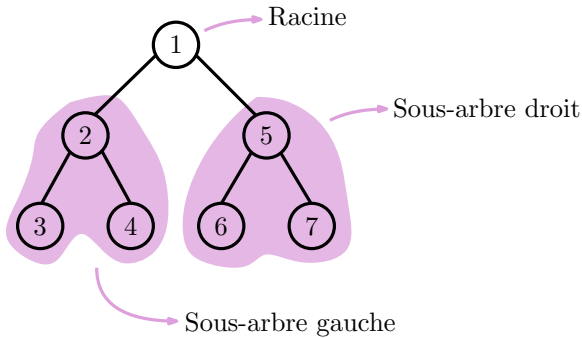
Un arbre binaire enraciné est défini récursivement comme étant soit

- Une feuille,
- Un sommet (la racine) à partir duquel pendent deux sous-arbres (les racines correspondantes sont les fils de gauche et de droite).

Definition (Arbre binaire)

Un arbre binaire enraciné est défini récursivement comme étant soit

- Une feuille,
- Un sommet (la racine) à partir duquel pendent deux sous-arbres (les racines correspondantes sont les fils de gauche et de droite).



En Racket, nous pouvons coder les arbres à l'aide de listes comme suit :

- Une feuille est représentée par une liste contenant 3 éléments : son étiquette et deux listes vides.

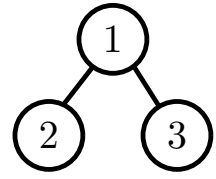

```
(0 () ())
```

Encodage - arbres binaires

En Racket, nous pouvons coder les arbres à l'aide de listes comme suit :

- Une feuille est représentée par une liste contenant 3 éléments : son étiquette et deux listes vides.
- Un sommet interne est représenté par une liste contenant 3 éléments : son étiquette et deux listes représentant chacune les sous-arbres gauche et droit.


```
(0 () ())
```

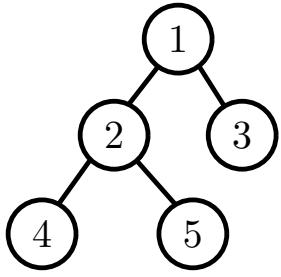



```
(1
  (2 () ())
  (3 () ()))
```

Encodage - arbres binaires

D'autres encodages sont possibles par exemple :

- Une feuille est représentée par une représentée par une seule valeur, l'étiquette.
- Un sommet interne est représenté par une liste contenant 3 éléments : son étiquette et les deux représentations des sous-arbres gauche et droit.




```
(1
  (2 3 4)
  3)
```

D'autres encodages sont possibles par exemple :

- Une feuille est représentée par une représentée par une seule valeur, l'étiquette.
- Un sommet interne est représenté par une liste contenant 3 éléments : son étiquette et les deux représentations des sous-arbres gauche et droit.

Remarquez que dans tous les cas, il suffit de définir une représentation pour les feuilles et pour les sommets internes : due la structure récursive des arbres, le reste est "automatiquement" défini !

D'autres encodages sont possibles par exemple :

- Une feuille est représentée par une représentée par une seule valeur, l'étiquette.
- Un sommet interne est représenté par une liste contenant 3 éléments : son étiquette et les deux représentations des sous-arbres gauche et droit.

Remarquez que dans tous les cas, il suffit de définir une représentation pour les feuilles et pour les sommets internes : due la structure récursive des arbres, le reste est "automatiquement" défini !
Nous nous en tiendrons à la première définition, car elle est uniforme et facile à manipuler.

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.
- `empty-tree?` qui renvoie `#t` si l'entrée représente un arbre vide et `#f` sinon.

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.
- `empty-tree?` qui renvoie `#t` si l'entrée représente un arbre vide et `#f` sinon.
- `leaf?` qui renvoie `#t` si l'entrée représente une feuille et `#f` sinon.

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.
- `empty-tree?` qui renvoie `#t` si l'entrée représente un arbre vide et `#f` sinon.
- `leaf?` qui renvoie `#t` si l'entrée représente une feuille et `#f` sinon.
- `tree=?` qui permet de tester si les deux arguments représentent des arbres identiques. `root` qui renvoie l'étiquette de la racine

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.
- `empty-tree?` qui renvoie `#t` si l'entrée représente un arbre vide et `#f` sinon.
- `leaf?` qui renvoie `#t` si l'entrée représente une feuille et `#f` sinon.
- `tree=?` qui permet de tester si les deux arguments représentent des arbres identiques. `root` qui renvoie l'étiquette de la racine
- left-subtree, right-subtree qui renvoie les représentations des sous-arbres gauche et droit respectivement.

Fonctions primitives sur les arbres

Nous allons maintenant construire une boîte à outils permettant de manipuler les arbres binaires. Elle est basée sur les fonctions primitives suivantes :

- `tree?` qui renvoie `#t` si l'entrée représente un arbre et `#f` sinon.
- `empty-tree?` qui renvoie `#t` si l'entrée représente un arbre vide et `#f` sinon.
- `leaf?` qui renvoie `#t` si l'entrée représente une feuille et `#f` sinon.
- `tree=?` qui permet de tester si les deux arguments représentent des arbres identiques. `root` qui renvoie l'étiquette de la racine
- left-subtree, right-subtree qui renvoie les représentations des sous-arbres gauche et droit respectivement.
- `tree` qui prend comme arguments une étiquette `v` et deux représentations d'arbres `l,r` et construit la représentation de l'arbre obtenu en reliant les racines des deux arbres représentés par `l,r` à une nouvelle racine étiquetée par `l`.

Fonctions primitives sur les arbres

```
(define (empty-tree? a)
  (null? a))
```

Fonctions primitives sur les arbres

```
(define (tree? a)
  (or (null? a)
      (and (= 3 (length a))
           (not (list? (car a)))
           (list? (cadr a))
           (tree? (cadr a))
           (list? (caddr a))
           (tree? (caddr a)))))
```

Fonctions primitives sur les arbres

```
(define (tree=? a b)
  (equal? a b))
```

Fonctions primitives sur les arbres

```
(define (root a)
  (car a))

(define (left-subtree a)
  (cadr a))

(define (right-subtree a)
  (caddr a))
```

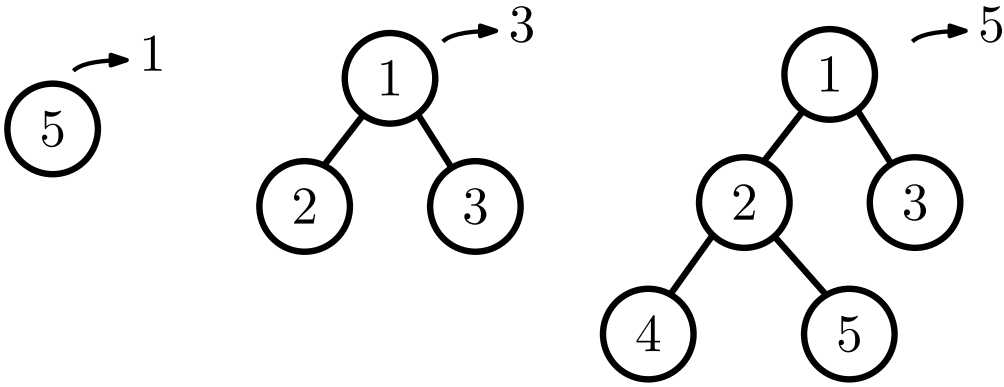
Fonctions primitives sur les arbres

```
(define (leaf? a)
  (and (not (empty-tree? a))
       (empty-tree? (fils-g a))
       (empty-tree? (fils-d a))))
```

Fonctions primitives sur les arbres

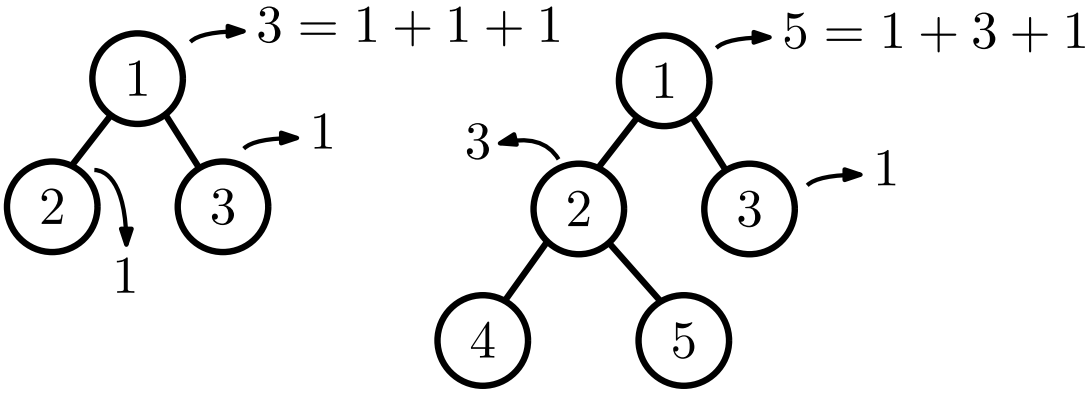
```
(define (leaf? a)
  (and (not (empty-tree? a))
       (empty-tree? (fils-g a))
       (empty-tree? (fils-d a))))
```

Les fonctions sur les arbres peuvent utiliser leur définition récursive naturelle. Par exemple, supposons que nous voulions écrire une fonction qui calcule le nombre de sommets d'un arbre :



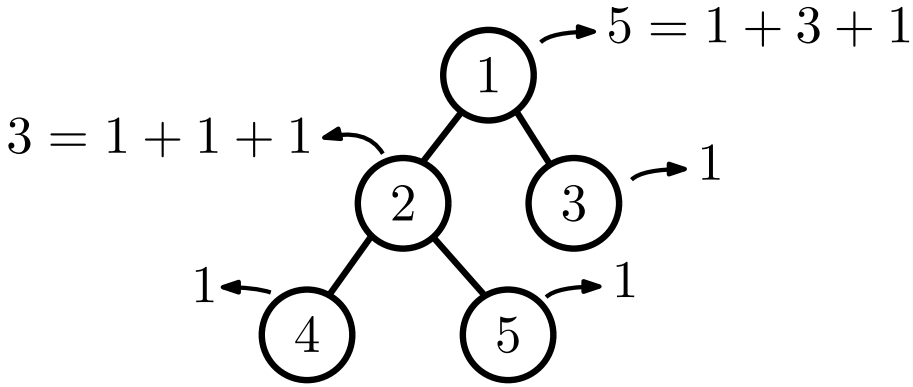
Fonctions recursives sur les arbres

Les fonctions sur les arbres peuvent utiliser leur définition récursive naturelle. Par exemple, supposons que nous voulions écrire une fonction qui calcule le nombre de sommets d'un arbre :



Fonctions recursives sur les arbres

Les fonctions sur les arbres peuvent utiliser leur définition récursive naturelle. Par exemple, supposons que nous voulions écrire une fonction qui calcule le nombre de sommets d'un arbre :



Les fonctions sur les arbres peuvent utiliser leur définition récursive naturelle. Par exemple, supposons que nous voulions écrire une fonction qui calcule le nombre de sommets d'un arbre :

```
(define (num-nodes t)
  (cond
    [(empty-tree? t) 0]
    [(leaf? t) 1]
    [else (+ (num-nodes (left-subtree t))
             (num-nodes (right-subtree t)))]))
```