(Séquence 8.1

Arbres généraux

Définition d'un arbre général

Dans un arbre général, chaque nœud porte une information (étiquette de type α) et a un nombre quelconque de descendants immédiats.
Le type est noté **ArbreGeneral(α)**

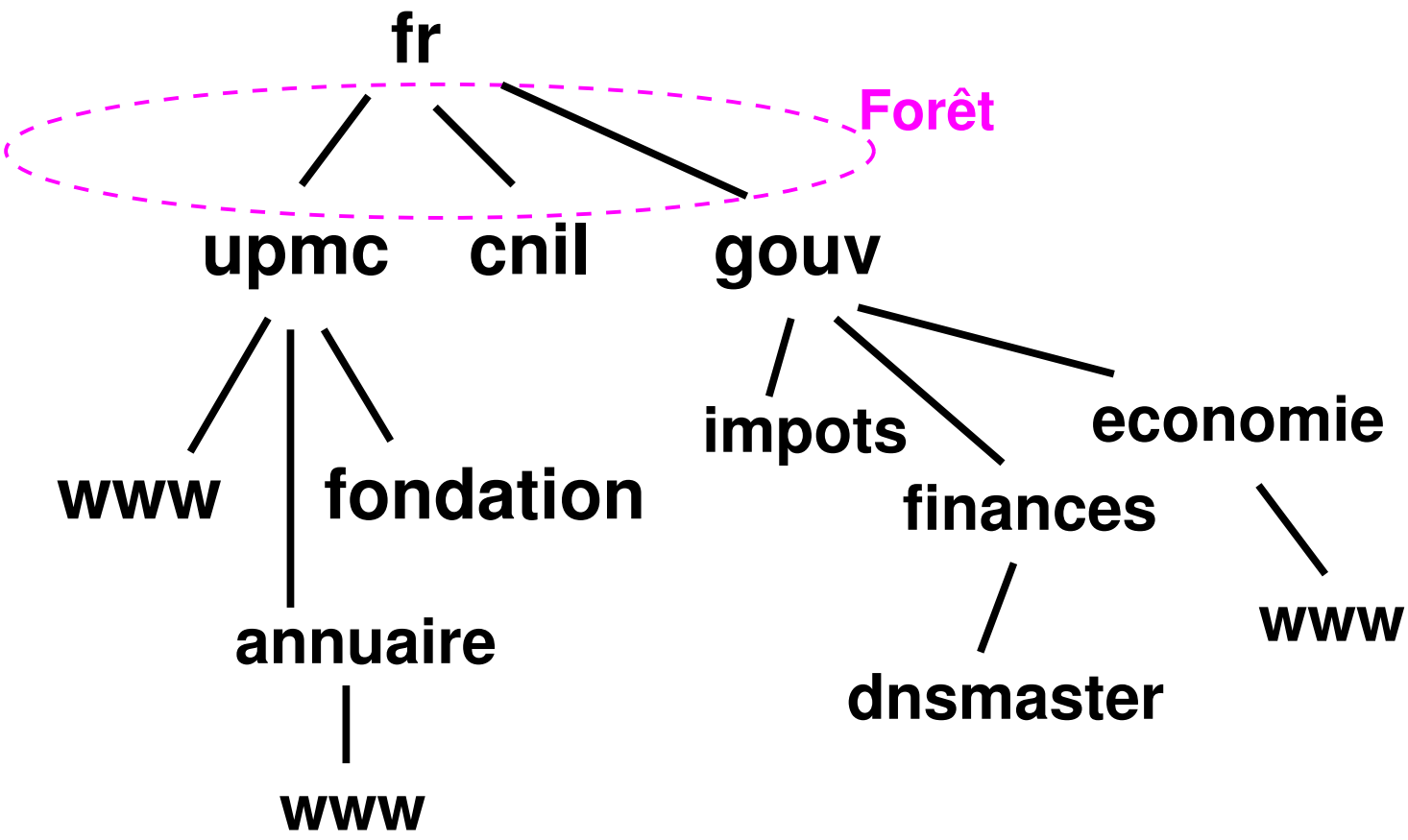
Définition d'un arbre général

Définition (récursive) Un arbre général est formé

- d'un nœud (portant une étiquette de type α)
- et d'une forêt (liste de sous-arbres) de type **LISTE(ArbreGeneral(α))**

Il n'y a pas d'arbre général vide, mais une forêt peut être vide

Exemple



Barrière d'abstraction des arbres généraux

- **Constructeur** pour construire un arbre général : `ag-noeud`
- **Accesseurs** pour accéder aux parties d'un arbre général : `ag-etiquette` et `ag-foret`
- **Reconnaisseur** inutile, puisqu'il n'y a qu'un seul constructeur

Remarque : les forêts sont des listes donc sont manipulées avec les primitives sur les listes

Spécification du constructeur

```
;;; ag-noeud: α * Foret[α] -> ArbreGeneral[α]
;;; avec Foret[α] = LISTE[ArbreGeneral[α]]
;;; (ag-noeud e foret) rend l'arbre formé de la
;;; racine d'étiquette e et, comme sous-arbres
;;; immédiats, les arbres de la forêt foret.
```

Exemples : arbres généraux de nombres

```
(ag-noeud 3 (list))    → #<object>
```

et construit l'arbre avec un unique nœud d'étiquette 3.

Spécification des accesseurs

```
;;; ag-etiquette: ArbreGeneral[α] -> α
;;; (ag-etiquette g) rend l'étiquette de la racine
;;; de l'arbre g.

;;; ag-foret: ArbreGeneral[α] -> Foret[α]
;;; avec Foret[α] = LISTE[ArbreGeneral[α]]
;;; (ag-foret g) rend la forêt des sous-arbres
;;; immédiats de g.
```

Propriétés algébriques

- Pour toute forêt d'arbres généraux F, et toute valeur v

```
(ag-etiquette (ag-noeud v F)) ≡ v
(ag-foret (ag-noeud v F))     ≡ F
```

- Pour tout arbre général G

```
(ag-noeud (ag-etiquette G)(ag-foret G)) ≡ G
```

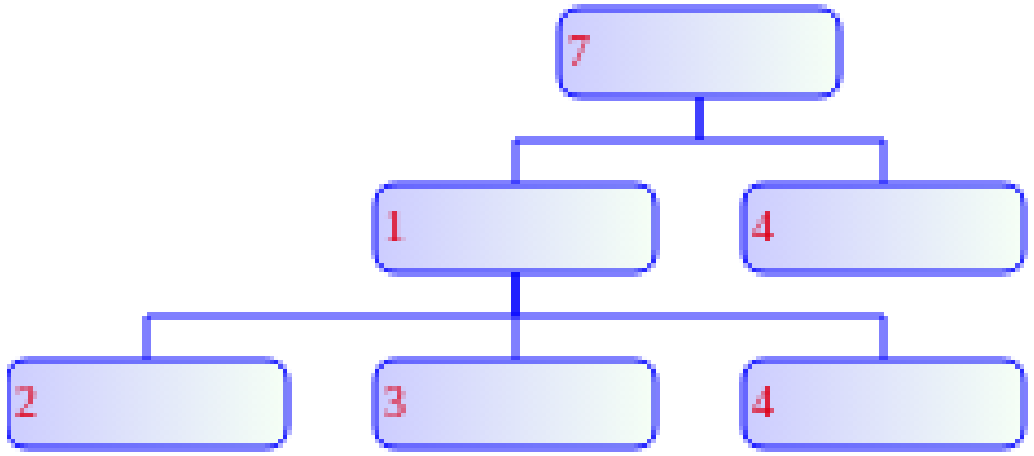
Reconnaisseur dérivé

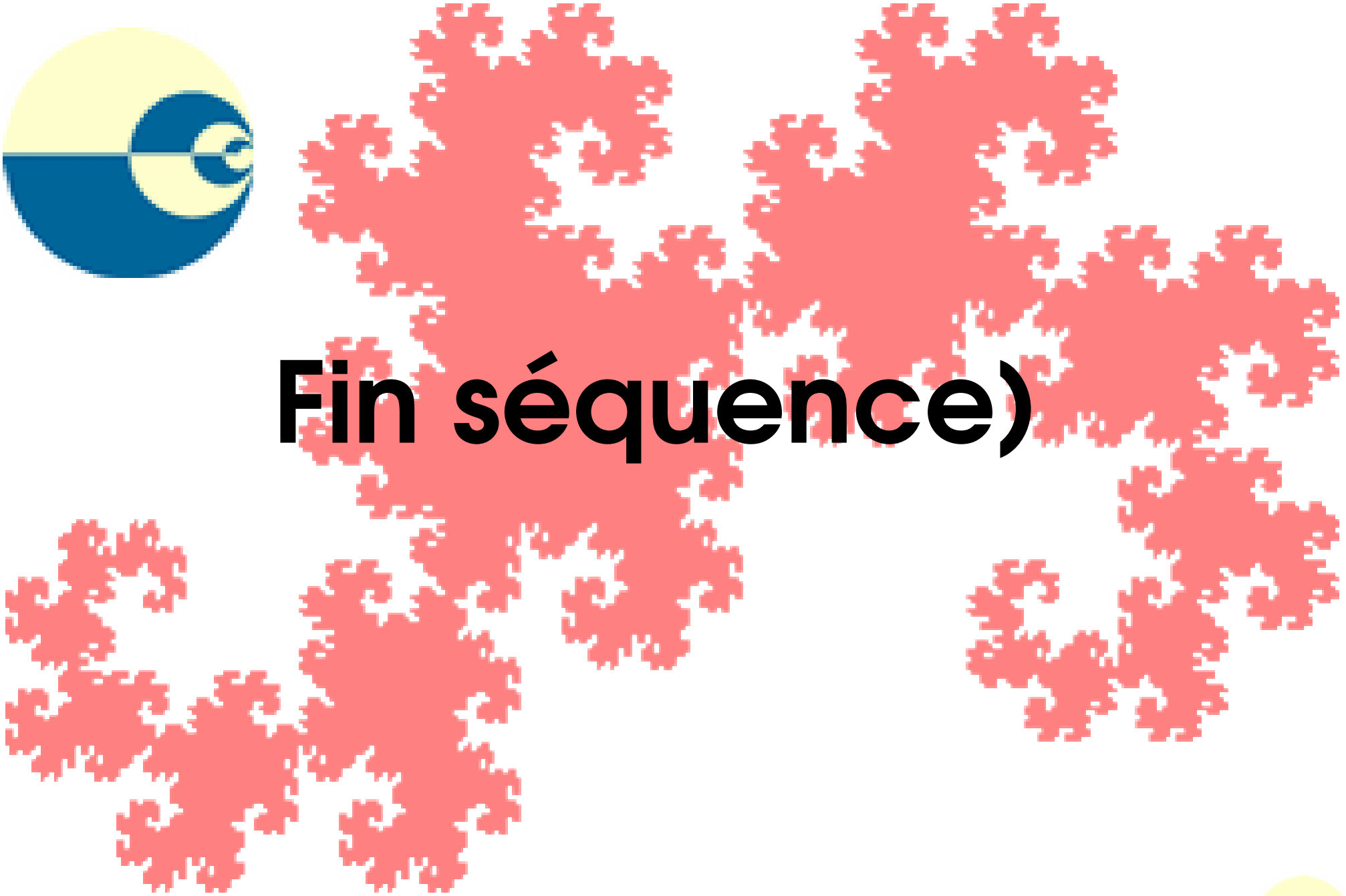
Définir le prédicat `ag-feuille?`

```
;;; ag-feuille?: ArbreGeneral[α] -> bool
;;; (ag-feuille? G) rend vrai ssi G est une feuille
(define (ag-feuille? G)
  (not (pair? (ag-foret G))) )
```

Une fonction d'affichage

```
(let* ((g1 (ag-noeud 2 '()))
       (g2 (ag-noeud 3 '()))
       (g3 (ag-noeud 4 '()))
       (g4 (ag-noeud 1 (list g1 g2 g3)))
       (g5 (ag-noeud 7 (list g4 g3))))
  (ag-affiche g5))
```





Fin séquence)