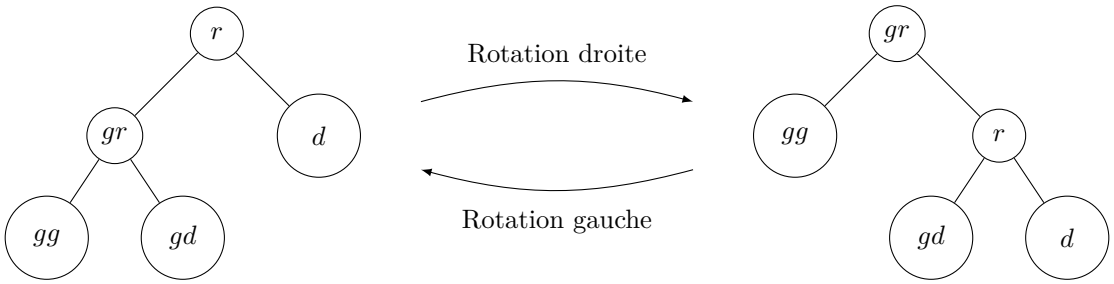
I Arbre AVL

Un arbre AVL est un ABR tel que, pour chaque noeud, la différence de hauteur de ses sous-arbres gauche et droit soit au plus 1. Pour éviter de calculer plusieurs fois la même hauteur, on stocke, dans chaque noeud, la hauteur de l'arbre correspondant. On utilise donc le type suivant: `type 'a avl = V | N of 'a * 'a avl * 'a avl * int`.

1. Montrer qu'un arbre AVL à n noeuds est de hauteur $O(\log(n))$.
   Indice : trouver une (in)équation de récurrence sur le nombre minimum de noeuds u_h d'un arbre AVL de hauteur h.
2. Écrire une fonction utilitaire `ht : 'a avl -> int` donnant la hauteur d'un AVL.
3. Écrire une fonction utilitaire `node : 'a -> 'a avl -> 'a avl -> 'a avl` construisant un AVL à partir d'une racine et de ses deux sous-arbres.

Lors de l'ajout d'un élément (en tant que feuille) la condition d'AVL peut être violée et doit être rétablie. Pour cela, on introduit l'opération de *rotation droite* (et son symétrique *rotation gauche*) autour d'un noeud:



4. Est-ce qu'une rotation préserve la propriété d'ABR?
5. Écrire une fonction `rotd` réalisant une rotation droite sur un arbre supposé de forme convenable.
   On suppose dans la suite avoir aussi une fonction `rotg` réalisant une rotation gauche (opération inverse).

On suppose que, après l'ajout d'un élément, `g` et `d` sont des AVL, mais que `N(r, g, d)` n'est pas un AVL. Pour les deux questions suivantes, on suppose que `ht g > ht d + 1`, l'autre cas étant symétrique. On décompose `g` en `N(gr, gg, gd)`.

6. Si `ht gg > ht gd`, montrer qu'une rotation suffit pour transformer `N(r, g, d)` en AVL.
7. Sinon, `ht gg < ht gd`. Montrer comment se ramener au cas précédent en une rotation.
8. En déduire une fonction `balance` prenant `r`, `g`, `d` en arguments et renvoyant l'AVL correspondant.
9. En déduire une fonction `add` ajoutant un élément dans un AVL en conservant la structure d'AVL.
10. Écrire aussi des fonctions `del` et `mem` pour supprimer un élément et savoir si un élément appartient à un AVL. On supposera qu'il n'y a pas de doublon dans l'AVL. Complexité de ces fonctions?

II Implémentation de dictionnaire avec arbre

On suppose avoir les fonctions suivantes sur des arbres binaires de recherche (qui peuvent être des AVL ou ARN, par exemple :

```
1    add : 'a -> 'a tree -> 'a tree
2    del : 'a -> 'a tree -> 'a tree
3    has : 'a -> 'a tree -> bool
```

En déduire des fonctions `get : 'a -> ('a * 'b) tree -> 'b option` et
`add : 'a * 'b -> ('a * 'b) tree -> ('a * 'b) tree` qui permettent d'implémenter un dictionnaire à l'aide d'un arbre, en mettant un couple (clé, valeur) sur chaque noeud (`'a` étant le type des clés et `'b` celui des valeurs).

III Problème de géométrie

On considère un ensemble E de n points dans le plan $\mathbb{R}^2$. Donner un algorithme pour trouver le nombre de rectangles que l'on peut former en choisissant 4 points de E. On pourra d'abord donner une solution simple en $O(n^4)$ puis une solution plus efficace, en utilisant un dictionnaire (quelle complexité obtient-on alors avec un dictionnaire implémenté par table de hachage? par arbre binaire de recherche équilibré ?).