Projet de Programmation Fonctionnelle
L2 Informatique, L3 Math-info, U. Bordeaux
Année 2016–17

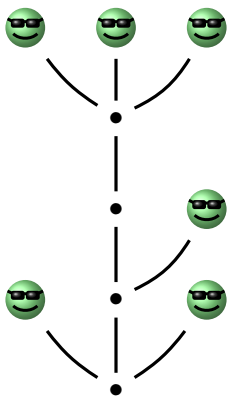
1 Modalités

Ce devoir est à rendre pour le **14 décembre 2016**, et à réaliser par groupe de 2 étudiant/e/s, ou 3 si les extensions proposées en fin de sujet sont réalisées. Le devoir doit être envoyé par mail à l'équipe pédagogique en respectant comme sujet du message : `DM-PF nom1 nom2`. Vos noms doivent aussi se trouver en commentaire dans le fichier `hydra_battle.ml`. Vous pouvez poser des questions par mail à l'équipe, nous répondons sur la liste de l'UE. Les fonctions demandées doivent avoir le nom imposé par le sujet. Certaines fonctions sont déjà fournies dans le fichier `hydra_battle.ml` de l'archive disponible sur le site web de l'UE, d'autres ne sont qu'un squelette indiquant leur nom et leur type. Vous pouvez bien sûr écrire des fonctions en plus de celles demandées (fonctions auxiliaires, fonctions de test, extensions, etc.).

2 Description du sujet

Le sujet s'articule autour du problème de batailles d'hydre, inspiré du deuxième des 12 travaux d'Hercule. Avant de lire le sujet, vous pouvez consulter cette courte vidéo, qui présente très clairement ce problème. Une description est également disponible ici (à partir du transparent 19). Enfin, on trouve sur le web des forums, ou des sites permettant de simuler une bataille d'hydre, avec des règles parfois différentes de celles présentées dans ce sujet, comme cette animation ou celle-ci.

Une hydre est un animal mythique (et également un animal réel) ayant plusieurs têtes, reliées à la base de l'hydre. On représente une hydre sous la forme d'un arbre fini, dont les nœuds peuvent avoir un nombre quelconque (mais fini) de filles. Les feuilles de l'arbre sont appelées les *têtes* de l'hydre. La base de l'hydre, représentée par la racine de l'arbre est appelée le *pied* de l'hydre. De chaque nœud interne de l'arbre partent des cous de l'hydre (au moins 1). La figure suivante représente une hydre à 6 têtes et 4 nœuds internes.

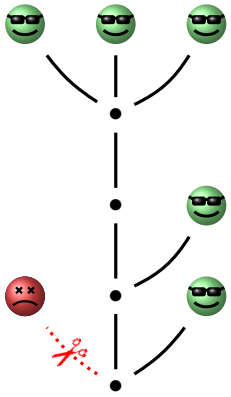


2.1 Les règles

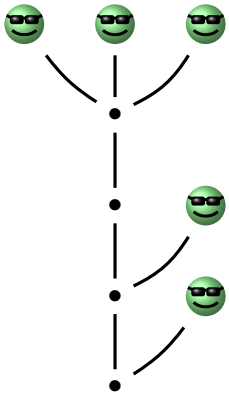
La bataille entre Hercule et l'hydre se déroule en tours, chacun effectué selon le modèle suivant :

1. Hercule choisit une **tête** de l'hydre et la coupe,
2. L'hydre réagit en se répliquant. Il y a 2 modes possibles de réplication :
   - le mode en surface,
   - le mode en profondeur.

Dans chacun des deux modes, si la tête coupée par Hercule est reliée directement au pied, alors l'hydre ne se réplique pas, et on passe au tour suivant : Hercule peut à nouveau couper une tête. C'est le cas sur la figure suivante.

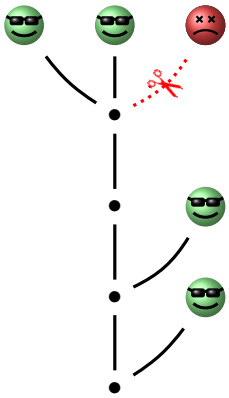


Choix d'une tête reliée au pied

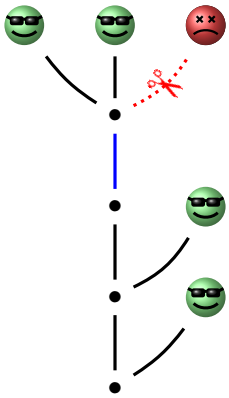


Résultat (pas de réplication)

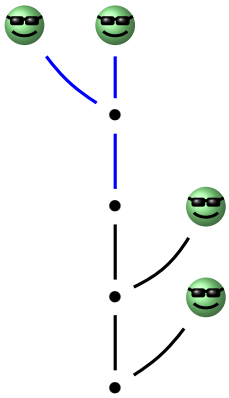
Le mode de réplication en surface est celui décrit dans la vidéo : lorsque la tête coupée n'est pas reliée au pied (étape 1), l'arête juste en dessous de celle qui a été coupée est marquée (étape 2) ainsi que toutes les arêtes au dessus de cette arête (étape 3). L'hydre a possibilité de répliquer cette partie, enracinée au même endroit, autant de fois qu'elle le souhaite (2 fois sur le dessin de l'étape 4, arêtes vertes).



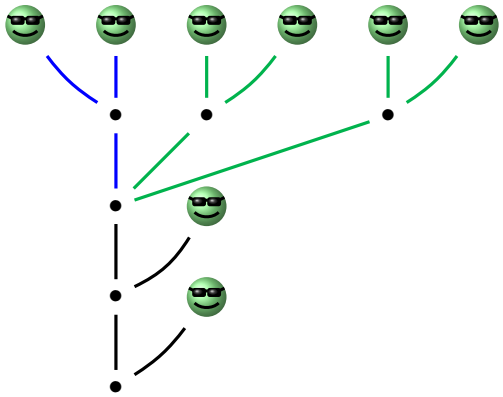
Étape 1



Étape 2

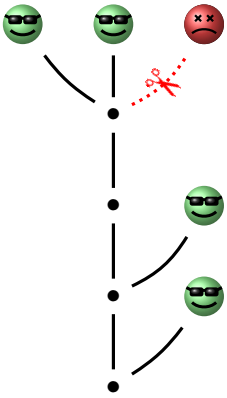


Étape 3

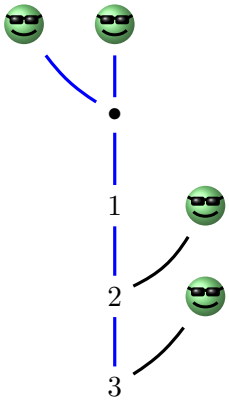


Étape 4 : réplication en surface

Pour l'autre mode de réplication, celui en profondeur, la règle change à l'étape 3 : l'hydre ne réplique pas seulement le sous-arbre identifié à l'étape 3. On marque aussi le chemin reliant la racine de ce sous-arbre à la racine, en numérotant les nœuds rencontrés sur ce chemin (à partir du haut). Par exemple, sur la même hydre que précédemment, si Hercule coupe la même tête, on obtient :

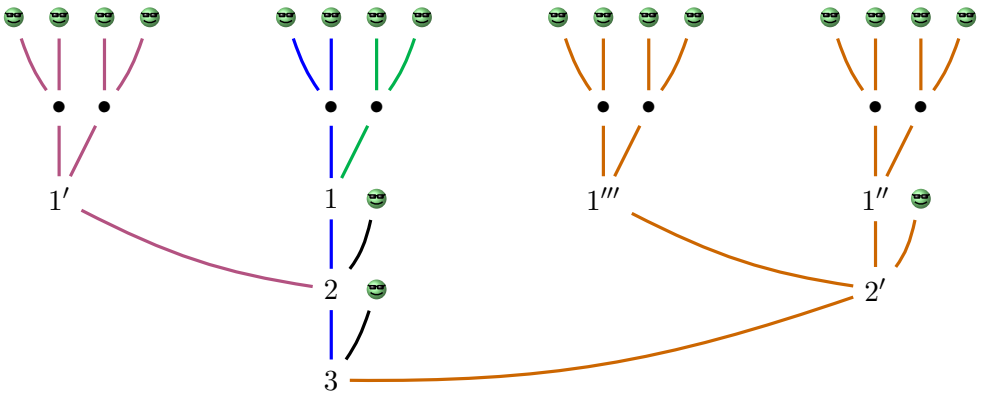


Hercule coupe une tête



Marquage du chemin de réplication

Au lieu de se faire seulement au nœud marqué 1 comme dans la réplication en surface, la réplication en profondeur s'effectue successivement aux nœuds 1, 2,… jusqu'à la racine de l'arbre.



Réplication en profondeur

Cette figure décrit la réplication en profondeur de l'hydre (ici, avec une seule copie à chaque niveau) :

- on effectue la réplication au nœud 1 comme dans la réplication en surface (en vert, ici une fois),
- le sous-arbre enraciné en 1 est ensuite répliqué au niveau du nœud 2 (en violet, ici une fois),
- enfin, le sous-arbre enraciné en 2 est répliqué au niveau du nœud 3 (en orange, ici une fois).

Le nombre de réplications est le même à chaque niveau. Il peut changer selon les tours : cela fait partie de la stratégie de l'hydre. Par exemple, l'hydre peut se répliquer k fois au $k^{\text{ème}}$ tour.

3 Travail demandé

Malgré la réplication de l'hydre, on peut montrer que *quels que soient ses choix*, Hercule finira par couper toutes les têtes. Il a aussi été prouvé qu'on ne peut pas démontrer ce résutat en utilisant l'arithmétique de Peano, qui est l'arithmétique habituelle sur les entiers, avec des axiomes comme le principe de récurrence (voir ici pour la preuve qu'Hercule finit toujours par gagner).

Même si Hercule finit toujours par gagner, le nombre de tours dépend bien sûr du nombre de fois que l'hydre va se répliquer, mais aussi des choix faits par Hercule. Dans le devoir, le type utilisé pour représenter l'hydre, des fonctions utiles ainsi que les fonctions de réplication de l'hydre sont fournies. Vous devrez compléter le fichier `hydra_battle.ml` fourni, plus précisément :

1. Écrire des fonctions donnant des informations sur les hydres (taille, hauteur, histogramme).
2. Écrire des fonctions permettant de visualiser les hydres sous forme graphique.
3. Programmer plusieurs stratégies d'Hercule.
4. Comparer ces stratégies.

La suite de cette section explique ces 4 points. On ne demande pas que les fonctions soient programmées de façon récursive terminale. Pour certaines fonctions, il faudra justifier leur correction et leur complexité.

Représentation des hydres

On représentera une hydre par le type suivant :

```
type hydra = Node of hydra list
```

Ce type a déjà été utilisé en TD (feuille sur les arbres). Pour construire des hydres, il est conseillé de développer des abréviations. Les suivantes sont par exemple fournies mais vous pouvez en ajouter :

```
(* Hydre à une seule tête *)
let head = Node []
(* Nœuds à 1, 2 ou 3 filles *)
let single h = Node [h]
let bi h1 h2 = Node [h1;h2]
let tri h1 h2 h3 = Node [h1;h2;h3]
(* Idem, lorsque les filles sont identiques *)
let bisame h = bi h h
let trisame h = tri h h h
```

Pour tester vos fonctions, plusieurs hydres sont prédéfinies dans le fichier `hydra_battle.ml`. Par exemple, la $1^{\text{ère}}$ hydre de ce document est `example_hydra`, celle obtenue par réplication en surface est `example_shallow`, et celle obtenue par réplication en profondeur est `example_deep`.

3.1 Informations sur les hydres

1. Écrire une fonction `size : hydra -> int` qui compte le nombre de nœuds (têtes et nœuds internes) d'une hydre. Par exemple, pour l'hydre en page 1, `size example_hydra` renverra `10`.
2. Écrire une fonction `height : hydra -> int` calculant la hauteur d'une hydre : la longueur d'un chemin maximal entre le pied et une tête. Par exemple, `height exemple_hydra` vaut `4`.
3. L'*histogramme* d'une hydre est par définition la liste du nombre de nœuds aux hauteurs 0, 1, 2, etc. Par exemple, l'histogramme de l'hydre `example_deep` est `[1; 3; 6; 8; 16]`. Écrire une fonction `histogram : hydra -> int list` qui calcule l'histogramme d'une hydre.
4. Écrire une fonction `histogram_heads : hydra -> int list` qui compte le nombre de têtes à chaque niveau. Par exemple, `histogram_heads example_deep` vaut `[0; 1; 2; 0; 16]`.

3.2 Visualisation des hydres

Pour visualiser les hydres, on se servira de l'utilitaire `dot` développé comme outil de la bibliothèque Graphviz. Cet outil permet de dessiner des graphes stockés dans des fichiers, avec une syntaxe simple. Dans ce devoir, vous n'avez pas besoin d'interagir avec `dot`, car une fonction `show_hydra` est déjà fournie : elle construit le graphe dans un fichier `tmp.dot`, le convertit en fichier image `tmp.png`, et lance les commandes Unix permettant d'afficher l'hydre. Si OCaml indique `Error : Reference to undefined global `Unix'`, ajoutez `#require "unix";;` dans le fichier `~/.ocamlinit`.

La fonction d'affichage appelle une fonction `hydra_edges : hydra -> (int * int) list` *non fournie* : l'objectif de cette partie est d'**écrire cette fonction**.

La fonction `hydra_edges` doit calculer une liste d'arêtes de l'hydre. Pour représenter une arête, on numérote chaque sommet par un entier dans $\mathbb{N}$, en respectant les contraintes suivantes :

- deux sommets différents n'ont pas le même numéro,
- le numéro d'un sommet est plus petit que les numéros attribués à ses filles,
- les numéros attribués aux filles d'un même sommet forment une suite croissante.

Une arête entre le sommet numéroté a et le sommet numéroté b, où $a < b$, sera alors représentée par le couple d'entiers $(a\,,b)$. Ainsi par exemple, `hydra_edges example_hydra` pourra s'évaluer en `[(0,1);(0,2);(0,3);(2,4);(2,5);(4,6);(6,7);(6,8);(6,9)]`.

Enfin, la liste renvoyée par la fonction `hydra_edges` doit être **triée** selon l'ordre lexicographique (c'est nécessaire pour que `dot` affiche les filles d'un nœud dans le bon ordre). Pour trier la liste, vous pouvez utiliser la fonction `List.sort` dont la description est disponible en ligne.

Indication

Pour obtenir la liste d'arêtes, un algorithme possible est le suivant : on retient à chaque étape le plus petit entier non encore utilisé pour numéroter un sommet (on l'appelle `next` dans la suite). On commence par numéroter le pied par 0 (`next` vaut alors 1). Une fois un nœud numéroté par un entier `k`, on numérote ses `n` filles par `next`, `next+1`,..., `next+n-1` et on ajoute donc à la liste d'arêtes (`k`, `next`), (`k`, `next+1`),...,(`k`, `next+n-1`). On recommence alors récursivement sur la liste des sous-arbres non traités. Si vous utilisez cet algorithme, vous pourrez écrire une fonction auxiliaire de type `(hydra * int) list -> int -> (int * int)`. Son premier argument est une liste d'hydres, chacune avec le numéro attribué à sa racine. Le second argument est la valeur de `next`. Le troisième argument est l'accumulateur qui mémorise les arêtes découvertes.

3.3 Stratégies d'Hercule

Une bataille entre Hercule et l'hydre est caractérisée par :

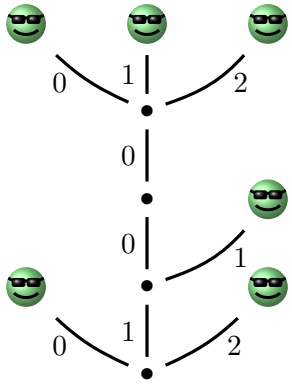
1. Le type de réplication de l'hydre : en surface ou en profondeur.
2. La stratégie de l'hydre.
3. La stratégie d'Hercule.

Les fonctions de réplication de l'hydre sont fournies :

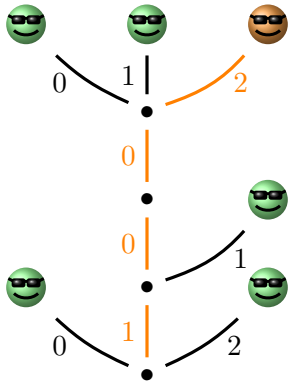
- `shallow_replication` pour la réplication en surface, et
- `deep_replication` pour la réplication en profondeur.

Une stratégie de l'hydre consiste simplement à choisir un nombre de réplications (au moins une) au tour k. Pour différencier clairement les entiers mesurant le temps et typer les fonctions, on a défini le type `time` par `type time = Time of int`. Une stratégie de l'hydre est donc donnée par une fonction `shy` de type `time -> int`, telle que `shy (Time k)` est le nombre de réplications que l'hydre choisit au tour k. Dans la réplication en profondeur, ce nombre est interprété comme le nombre de réplications à chaque niveau le long du chemin de réplication. Ces stratégies sont simples à concevoir, deux vous sont fournies (`original_hydra_strat` et `boum`).

On demande de programmer plusieurs stratégies d'Hercule. Une stratégie d'Hercule consiste à choisir une tête de l'hydre. On désignera une tête par le chemin qui va du pied à la tête. Pour le représenter, on numérote les filles de chaque nœud en partant de 0, comme sur la figure suivante.



Un chemin est alors la suite d'entiers utilisés depuis le pied lorsqu'on parcourt le chemin. Par exemple, la tête la plus à droite parmi les 3 têtes de hauteur 4 est désignée par le chemin pour y accéder : `[1;0;0;2]`.



Le chemin `[1;0;0;2]` qui désigne une tête

On définit donc le type chemin par

```
type path = int list
```

et une stratégie d'Hercule est donc du type :

```
type hercules_strat =  hydra -> path
```

Notez qu'une liste d'entiers peut ne pas désigner un chemin valide allant jusqu'à une tête. On demande d'écrire les fonctions suivantes :

1. `check_hercules_strategy : hercules_strat -> hydra -> bool`, de telle sorte que l'appel `check_hercules_strategy s h` renvoie `true` ssi la stratégie `s` sélectionne une tête de `h`.
2. `leftmost_head_strat : hercules_strat`, telle que `leftmost_head_strat h` sélectionne la tête la plus à gauche dans l'hydre `h`, c'est-à-dire fournit un chemin maximal du type `[0;0;...;0]`.
3. `highest_head_strat : hercules_strat`, qui sélectionne une tête de hauteur maximale.
4. `closest_to_ground_strat : hercules_strat` qui sélectionne une tête de hauteur minimale.
5. `random_strat : hercules_strat` qui sélectionne une des têtes au hasard.

Pour la stratégie `leftmost_head_strat`, on demande aussi :

6. de justifier (en commentaire dans le fichier) que votre fonction a bien le comportement demandé, en utilisant une récurrence.
7. d'évaluer la complexité de votre fonction, en justifiant.

3.4 Simulations et comparaison de stratégies

Une fois les stratégies écrites, on peut simuler une bataille, déterminée par son genre : la fonction de réplication (en surface ou en profondeur), la stratégie d'Hercule et la stratégie de l'hydre.

```
type genre_de_bataille = Battle_kind of replication_fun * hercules_strat * hydra_strat
```

Le résultat d'une bataille lorsqu'un nombre de tours k a été fixé est

- soit une victoire d'Hercule, en un certain nombre de tours inférieur ou égal à k,
- soit un abandon d'Hercule, avec une hydre restante qui est plus grande qu'une unique tête.

Un tel résultat est représenté par :

```
 type result =
  Hercules_wins of time       (* Nombre de tours effectués *)
| Hercules_gives_up of hydra  (* Hydre restante, plus qu'une seule tête *)
```

1. Écrire une fonction `simulation : genre_de_bataille -> hydra -> time -> result` telle que `simulation b h (Time t)` donne le résultat de la bataille de type `b` partant de l'hydre `h` pendant un temps `t`.
2. On appelle *mesure* une fonction qui à une hydre associe une valeur. Une mesure est donc de type `hydra -> 'a`. Par exemple, la taille ou l'histogramme sont des mesures. Écrire une fonction la plus efficace possible

   ```
   make_trace : (hydra -> 'a) -> genre_de_bataille -> hydra -> time -> 'a list
   ```

   telle que `make_trace measure bat h_init (Time t)` donne la suite des valeurs de la fonction `measure` sur les hydres obtenues en partant de l'hydre `h_init` et en effectuant `t` tours de la bataille de genre `bat`.
   Ainsi par exemple, en appliquant `make_trace` avec comme premier argument la fonction `size`, on obtient la liste donnant l'évolution de la taille de l'hydre pendant la bataille.

4 Extensions possibles

Si le temps le permet, vous pouvez réaliser des extensions, comme :

- prouver qu'une stratégie d'Hercule que vous avez écrite autre que la stratégie choisissant la tête la plus à gauche sélectionne toujours une tête.
- réfléchir à un moyen de comparer deux hydres, et si possible l'implémenter. En particulier, on considérera que deux hydres sont identiques si elles ne diffèrent que par commutation de certains de leurs cous.
- écrire des fonctions de génération d'hydres « aléatoires », en fixant certains paramètres, comme la taille ou la hauteur, et en réfléchissant à la probabilité de générer une hydre donnée.