TP 3 – Applications

1 Mutants

On considère la base de faits Prolog suivante :

```
animal(hibou).          animal(caribou).     animal(bouquetin).
animal(ours).           animal(tigre).       animal(alligator).
animal(grenouille).     animal(singe).       animal(hippopotame).
```

etc. avec `ouistiti`, `crabe`, `becasse`, `tortue`, `vache`, `chevre`, `cheval`, `lapin`, `pelican`, `marmotte`, `canard`, `lama`, `tetard`, `pintade`, `truite`, et d'autres.

Un mutant de première génération est un animal dont le nom (un atome) commence par le nom d'un animal, finit par le nom d'un autre tel que ce dernier commence comme finit le premier ... Ça n'est pas clair ? Exemple `caribouquetin` est un mutant descendant d'un `caribou` et d'un `bouquetin`.

1. Écrire un prédicat de nom `mutant/1` réussissant si son argument est un mutant de première génération. Se servir de ce prédicat pour afficher tous les mutants de première génération.
2. Écrire un prédicat de nom `gmutant/2` tel que `gmutant(M, Gen)` réussisse en instanciant `M` avec un mutant de `Gen`ième génération. Remarque : un mutant de génération 0 est un animal dont le nom est présent dans la base de fait ; Un mutant de génération n est le fils d'un mutant de génération n_1 et d'un mutant de génération n_2 tel que $n = n_1 + n_2$.
3. Généraliser avec le prédicat de nom `gmutant/1`, cherchant les mutants de génération quelconque.

2 Crypto-arithmétique

Une opération crypto-arithmétique est une opération où les chiffres des nombres en jeu sont remplacés par des lettres suivant le principe que deux lettres différentes concernent toujours deux chiffres différents. Un exemple célèbre est l'addition suivante :

```
  SEND
  MORE
 -----
 MONEY
```

Résoudre ce problème en utilisant Prolog.

Réaliser un autre programme qui résout la crypto-addition suivante :

```
   TEN
   TEN
 FORTY
 -----
 SIXTY
```