# CR - TME 5

# Exercice 1

- `subs(chat,felin).` $chat \sqsubseteq felin$ Les chats sont des felins
- `subs(chihuahua,and(chien,pet)).` $chihuahua \sqsubseteq chien \wedge pet$ Un chihuahua est à la fois un chien et un animal de compagnie
- `subs(and(animal,some(aMaitre)),pet).` $animal \wedge \exists aMaitre \sqsubseteq pet$ Un animal qui a un maˆıtre est un animal de compagnie
- `subs(some(aMaitre),all(aMaitre,personne)).` $\exists aMaitre \sqsubseteq \forall aMaitre.personne$ Toute entité qui a maître ne peut avoir qu'un (ou plusieurs) maître(s) humain(s)
- `subs(and(all(mange,nothing),some(mange)),nothing).` $\forall mange.\bot \wedge \exists mange \sqsubseteq \bot$ On ne peut pas à la fois ne rien manger (ne manger que des choses qui n'existent pas) et manger quelque chose.
- `equiv(carnivoreExc,all(mange,animal)).` $carnivoreExc \equiv \forall mange.animal$ Un carnivore exclusif est défini comme une entité qui mange uniquement des animaux

## Excercice 2

### Question 1

- `subsS1(C,C).` traduit l'égalité de l'inclusion $\sqsubseteq$
- `subsS1(C,D):-subs(C,D),C\==D.` traduit l'inclusion sans égalité $\sqsubset$
- `subsS1(C,D):-subs(C,E),subsS1(E,D).` traduit le fait qu'il faut fouiller dans chaque subs, en passant par une récursion.

```
?- subsS1(canari, animal).
true ;
true ;
false.

?- subsS1(chat,etreVivant).
true ;
true ;
false.
```

La récursion fonctionne bien.

### Question 2

```
?- subsS1(chien,souris).
false.
```

Cette requête provoque une boucle infini. Prolog fait

- Chien - canidé - mamifaire - etreVivant => X donc ça reboucle pour chercher un autre chemin
- Chien - canidé - chien => La boucle est bouclé

## Question 3

La trace est grande. On voit que la liste de variable déjà checké augmente bien.

## Question 4

`subsS(souris, some(mange)).` `true` il passe par $souris \sqsubset animal \sqsubset \exists mange$.

## Question 5

`subsS(chat, X).` devrait renvoyé tous les parents de chat, en remontant l'arbre.

```
?- subsS(chat, X).
X = chat ;
X = felin ;
X = mammifere ;
X = animal ;
X = etreVivant ;
X = some(mange) ;
false.
```

`subsS(X, mammifere).` devrait renvoyé tous les enfants de mammifère, en descendant l'arbre.

```
?- subsS(X, mammifere).
X = mammifere ;
X = felin ;
X = canide ;
X = souris ;
X = chat ;
X = chien ;
X = lion ;
false.
```

## Question 6

Avant l'ajout des règles d'équivalence.

```
?- subsS(lion, all(mange,animal)).
false.
```

Après l'ajout des règle suivante :

```
subs(C,D) :- equiv(C,D).
subs(D,C) :- equiv(C,D).
```

```
?- subsS(lion, all(mange,animal)).
true ;
true ;
```

```
true ;
true ;
true ;
true ;
true ;
true ;
false.
```

Je ne sais pas trop ce qu'il fait, d'après le prof, on ne l'a pas assez limité dans sa recherche, du coup il trouve plusieurs chemin possible.

# Exercice 3

## Question 1

- `subsS(chihuahua, and(mammifere, some(aMaitre)))`
- `subsS(and(chien, some(aMaitre)), pet)`
- `subsS(chihuahua, and(pet,chien))`

## Question 2

- `subsS(C,and(D1,D2),L) :- D1\=D2, subsS(C, D1, L), subsS(C, D2, L).` l'instruction de base pour les intersection, on regarde si il est dans l'un ou dans l'autre. Sans cette règles, pas d'intersection, je pense qu'on ne saurait pas traiter les `and()`.
- `subsS(C, D, L) :- subs(and(D1, D2), D), E=and(D1, D2), not(member(E, L)), subsS(C, E, [E|L]), E\==C.`. Ici on regarde si `C` ne peux pas se décomposer en une intersection `E` dans laquelle on a pas encore chercher récursivement. Puis on regarde si `C` n'est pas dans cette éventuelle décomposition. Pourquoi ???
- `subsS(and(C, C), D, L) :- nonvar(C), subsS(C, D, [C|L]).` Si c'est n'est pas une variable, alors on cherche `subsS`
- `subsS(and(C1, C2), D, L) :- C1\=C2, subsS(C1, D, [C1|L]).` Si on peut décomposer `C` en une intersection de `C1` et `C2` alors on vas chercher récursivement dans ces deux ensemble.
- `subsS(and(C1, C2), D, L) :- C1\=C2, subsS(C2, D, [C2|L]).` Same
- `subsS(and(C1, C2), D, L) :- subs(C1, E1), E=and(E1, C2), not(member(E, L)), subsS(E, D, [E|L]), E\==D.` On regarde si il n'y a pas un autre ensemble permetant de relier `C1` et `C2`.
- `subsS(and(C1, C2), D, L) :- Cinv=and(C2, C1), not(member(Cinv, L)), subsS(Cinv, D, [Cinv|L]).` Prise en compte du `and()` avec les paramètre permutable.

# Exercice 4

# Question 1

impossible, au secours, j'abandonne.

J'ai principalement tenté d'implémenter une règle type $C \subseteq D \Leftrightarrow R.C \subseteq R.D$ sans avoir de boucle infini. J'ai réussi une fois pour la query 'lion' mais ça ne fonctionnait pas pour celle du

canari. Puis j'ai changé des truc et ça plus rien ne fonctionnait, impossible de revenir à l'état d'avant où la première fonctionnait pfffff

```
subsS(C,D, L) :- subsS(all(R,C), all(R,D), L), E=all(R,D), not(member(E,L)), subsS(E, D, [E|L]), E\==D.
```