Énoncé

1 Écrire un programme Prolog, avec le prédicat `est_un/2`, qui asserte les faits suivants :

Un loup est un mamifère
Un chat est un mamifère
Une fourmi est un insecte
Un mamifère est un vertébré
Un insecte est un invertébré
Un vertébré est un animal
Un invertébré est un animal

2 Définir le prédicat `même_genre/2` qui est vrai de deux classes qui ont la même surclasse. [Rappel : on note `\=` l'inégalité de termes.]

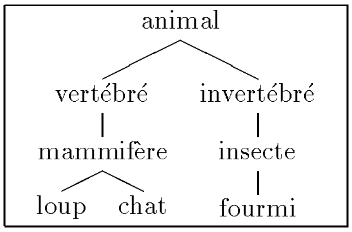
3 Dessiner l'arbre de recherche (complet) correspondant à la question :
`?- même_genre(loup,X).`

4 Définir le prédicat `surclasse_commune/2` qui est vrai de deux classes qui ont une surclasse (immédiate ou non) commune. [Remarque : Ce prédicat est défini de manière récursive, par l'intermédiaire de plusieurs clauses.]

5 Définir un prédicat `surclasses/2` qui construit la *liste* des surclasses d'une classe donnée. On admettra que l'on dispose d'un opérateur, noté `\+`, tel que le but `\+ B` s'interprète comme « `B` n'est pas prouvable ».

Corrigé

1

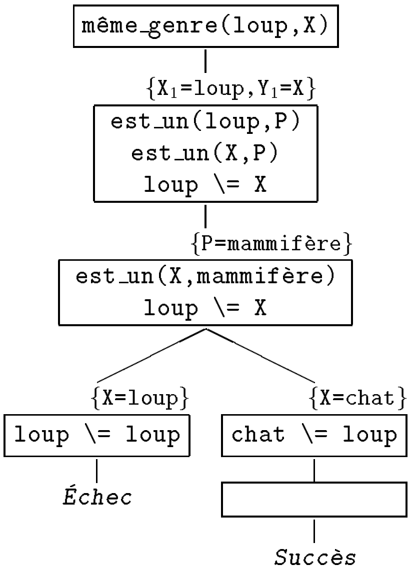



```
est_un(loup,mammifere).
est_un(chat,mammifere).
est_un(fourmi,insecte).
est_un(mammifere,vertebre).
est_un(insecte,invertebre).
est_un(vertebre,animal).
est_un(invertebre,animal).
```

2

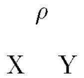
```
meme_genre(X,Y) :-
        est_un(X,P),
        est_un(Y,P),
        X \= Y.
```

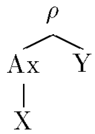
3

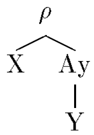


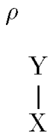
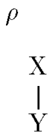
4

Il est naturel de tenter de ré-utiliser le prédicat déjà défini `même_genre/2`, comme condition d'arrêt. Dans ce cas, compte tenu de la définition de ce prédicat, qui est telle qu'on n'a pas `meme_genre(X,X)`, il faut prendre en considération 3 cas différents (et les symétriques correspondant).

`surclasse_commune(X,Y) :-` `    meme_genre(X,Y)`	

`surclasse_commune(X,Y) :-`
`    est_un(X,Ax),`
`    surclasse_commune(Ax,Y).`
	

`surclasse_commune(X,Y) :-`
`    est_un(Y,Ay),`
`    surclasse_commune(X,Ay).`
	

`surclasse_commune(X,Y) :-` `    herite(X,Y).`	
`surclasse_commune(X,Y) :-` `    herite(Y,X).`	

avec la définition suivante du prédicat `herite/2`:

```
herite(X,Y) :- est_un(X,Y).
herite(X,Y) :- est_un(X,Ax), herite(Ax,Y).
```

On peut aussi proposer, en reconsidérant tout le problème, et en admettant que la relation définie par le prédicat soit réflexive (X et lui-même ont une surclasse commune), la forme suivante :

```
surclasse_commune(X,X).
surclasse_commune(X,Y) :- est_un(X,Ax), surclasse_commune(Ax,Y).
surclasse_commune(X,Y) :- est_un(Y,Ay), surclasse_commune(X,Ay).
```

À noter que dans les deux cas, ces prédicats ne fonctionnent pas de manière satisfaisante si on leur fait faire du retour-arrière (redondance des solutions trouvées). Pour éviter cet inconvénient, on peut aussi reformuler le problème en disant que deux éléments ont une surclasse commune si ils sont dans le même arbre, c'est-à-dire s'ils ont la même racine, ou encore si la racine de l'un est une surclasse de l'autre. La mise en œuvre d'une telle solution nécessite cependant un opérateur de coupure (« *cut* ») :

```
% Y est la racine de X
racine(X, Y) :- est_un(X, Z), !, racine(Z, Y).
racine(X, X).

% X a pour surclasse Y
sur_classe0(X, X).
sur_classe0(X, Y) :- est_un(Z, X), sur_classe0(Z, Y).

scc(X, Y) :- racine(X, Z), sur_classe0(Z, Y).
```

5

```
surclasses(X,[T|Q]) :-
        est_un(X,T),
        surclasses(T,Q).
surclasses(X,[]) :-
        \+ est_un(X,_).
```