Introduction à l'IA symbolique — TPs

Ivan Lejeune

10 décembre 2025

Table des matières

TP1 — Introduction à Clingo

Exercice 1.1 Pour commencer en logique des propositions.
Commençons par un exemple simple en logique des propositions (tous les symboles devront commencer par une minuscule puisqu'il n'y a pas de variable) : Benoît et Cloé une réunion d'amis ; Cloé veut absolument inviter Djamel ; si Félix vient, il faut inviter Amandine ; si Emma vient, il faut inviter Xéna ; Benoît voudrait inviter Félix et Xéna, mais Xéna ne supporte pas Amandine, donc Benoît n'invite Xéna que si Amandine ne vient pas. On modélise cette situation en utilisant des symboles propositionnels associés à l'initiale de chaque prénom (par exemple, le symbole b signifie « Benoît vient »). On obtient 2 faits (b et c) et 5 règles :

- Si c alors d (ou plutôt « d si c », ce qu'on note $d :\!= c$),
- Si f alors a,
- Si e alors x,
- Si b alors f,
- Si b et not a alors x.

Qui viendra à la réunion. La base de connaissances est satisfiable (ouf !) et Clingo fait remarquer au passage qu'Emme ne sera de toute façon pas invitée, car e n'apparaît dans aucune tête de règle, c'est-à-dire ni dans un fait — vu comme une règle à corps vide — ni dans une conclusion de « vraie » règle.

Solution.
On se sert de l'outil Clingo présenté dans le TP pour modéliser le problème.

```
% faits connus
b. % benoit vient
c. % cloe vient

% regles
d :- c.
a :- f.
x :- e.
f :- b.
x :- b, not a.
```

On lance Clingo sur ce programme et on obtient la sortie suivante :

```
Answer: 1
c d f a b
SATISFIABLE
```

On constate que la base de connaissances est satisfiable et que les invités sont Cloé, Djamel, Félix, Amandine et Benoît.

Exercice 1.2 Pour commencer en logique du premier ordre.
Passons à la logique du premier ordre. En réalité, Clingo instancie chaque règle par toutes les constantes apparaissant dans la base de connaissances (mais avec plus de discernement que la méthode brutale vue en cours), il se ramène donc à la logique des propositions, même si l'utilisateur ne le voit pas

1. Modéliser les connaissances suivantes (en 3 faits et 8 règles) en utilisant les prédicats unaires **animal**, **plante**, **carnivore**, **herbivore**, **omnivore** et **humain**, et le prédicat binaire **mange** (« x mange y ») :
   (a) La chèvre de Monsieur Seguin est un herbivore.
   (b) Le loup de Monsieur Seguin est un carnivore.
   (c) Le petit chaperon rouge est un humain.

(d) Les carnivores et les herbivores sont des animaux.

(e) Les omnivores sont à la fois des carnivores et des herbivores (« si on est un omnivore alors on est un carnivore et un herbivore »).

(f) Les humains sont des omnivores.

(g) Tout animal carnivore ne mange que des animaux.

(h) Tout animal herbivore ne mange que des plantes.

(i) Tout animal carnivore mange n'importe quel animal herbivore.

Vous constaterez que Clingo produit des faits étranges : le petit chaperon rouge se mange lui-même, la chèvre est une plante, ainsi que le petit chaperon rouge.

2. On modifie la phrase 9 : Tout animal carnivore mange n'importe quel animal herbivore **différemment de lui-même**. Prendre en compte cette modification : le symbole de différence est `!=` et c'est une macro pour `not ==`, autrement dit $(X \neq Y)$ est vrai si rien n'indique que X est égal à Y. Le petit chaperon rouge devrait aller mieux.

3. Nous savons que les animaux ne sont pas des plantes (ou l'inverse), autrement dit les deux ensembles d'entités sont disjoints. Ceci peut s'exprimer sous la forme d'une règle appelée « **contrainte négative** » :

$$\forall X.\ \text{animal}(X) \wedge \text{plante}(X) \rightarrow \bot$$

Avec la syntaxe de Clingo, ceci s'écrit comme une règle à tête vide (un corps vide est considéré comme toujours vrai, une tête vide comme toujours fausse) :

```
:- animal(X), plante(X). % contrainte negative
```

Ajouter cette contrainte. Que répond Clingo.

4. Bien sûr, c'est la définition d'omnivore qui ne convient pas : un omnivore n'est ni un carnivore ni un herbivore. Commenter les règles qui définissent omnivore, et indiquer simplement qu'un omnivore est un animal. Exécuter à nouveau Clingo et analyser le résultat.

Solution.

1. On modélise les connaissances en Clingo comme suit :

```
% faits connus
herbivore(chevre). % (a)
carnivore(loup).   % (b)
humain(chaperon).  % (c)

% regles
animal(X) :- carnivore(X).   % (d1)
animal(X) :- herbivore(X).   % (d2)

carnivore(X) :- omnivore(X). % (e1)
herbivore(X) :- omnivore(X). % (e2)

omnivore(X) :- humain(X).    % (f)

animal(Y) :- animal(X), carnivore(X), mange(X,Y). % (g)
plante(Y) :- animal(X), herbivore(X), mange(X,Y). % (h)

mange(X,Y) :- animal(X), animal(Y), carnivore(X), herbivore(Y). % (i)
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
carnivore(loup) carnivore(chaperon) animal(loup) animal(chaperon) animal(chevre)
```

```
herbivore(chevre) herbivore(chaperon) omnivore(chaperon) humain(chaperon)
mange(loup,chevre) mange(chaperon,chevre) mange(loup,chaperon)
mange(chaperon,chaperon) plante(chevre) plante(chaperon)
SATISFIABLE
```

En effet, on a quelques résultats étranges, comme le petit chaperon rouge qui se mange lui-même.

2. On modélise la modification de la phrase 9 en Clingo comme suit :

```
mange(X,Y) :- animal(X), animal(Y), carnivore(X), herbivore(Y), X!=Y. % (i)
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
carnivore(loup) carnivore(chaperon) animal(loup) animal(chaperon) animal(chevre)
herbivore(chevre) herbivore(chaperon) omnivore(chaperon) humain(chaperon)
mange(loup,chevre) mange(chaperon,chevre) mange(loup,chaperon) plante(chevre)
SATISFIABLE
```

C'est plus raisonnable.

3. On ajoute la contrainte négative dans le fichier Clingo :

```
:- animal(X), plante(X). % (j) -- contrainte negative
```

En lançant Clingo, on obtient la sortie suivante :

```
UNSATISFIABLE
```

La base de connaissances est insatisfiable, car on a par exemple la chèvre qui est à la fois un animal et une plante.

4. On modélise la nouvelle définition d'omnivore en Clingo comme suit :

```
% carnivore(X) :- omnivore(X). % (e1)
% herbivore(X) :- omnivore(X). % (e2)
animal(X) :- omnivore(X). % (k / e3)
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
carnivore(loup) animal(loup) animal(chevre) animal(chaperon)
herbivore(chevre) omnivore(chaperon) humain(chaperon) mange(loup,chevre)
SATISFIABLE
```

La base de connaissances est maintenant satisfiable, et les résultats sont plus cohérents.

Le code final est le suivant :

```
% faits connus
herbivore(chevre). % (a)
carnivore(loup).   % (b)
humain(chaperon).  % (c)

% regles
animal(X) :- carnivore(X).   % (d1)
animal(X) :- herbivore(X).   % (d2)

% carnivore(X) :- omnivore(X). % (e1)
% herbivore(X) :- omnivore(X). % (e2)
animal(X) :- omnivore(X). % (k / e3)

omnivore(X) :- humain(X).    % (f)
```

```
animal(Y) :- animal(X), carnivore(X), mange(X,Y). % (g)
plante(Y) :- animal(X), herbivore(X), mange(X,Y). % (h)

mange(X,Y) :- animal(X), animal(Y), carnivore(X), herbivore(Y), X!=Y. % (i)

:- animal(X), plante(X). % (j) -- contrainte negative
```

Exercice 1.3 La famille d'Oedipe.
Le fichier oedipe-family-facts.lp (voir moodle) fournit une base de faits décrivant la famille d'Oedipe, personnage de la mythologie grecque. Vous pouvez copier-coller ce fichier dans la fenêtre de Clingo. Les prédicats utilisés sont les suivants (où / indique l'arité du prédicat) : personnage/1, homme/1, femme/1, aEnfant/2 (qui lie un parent à l'un de ses enfants), roi/2 (qui lie un roi à la ville dont il est roi). La figure ci- dessous donne une vue d'ensemble de cette base de faits :

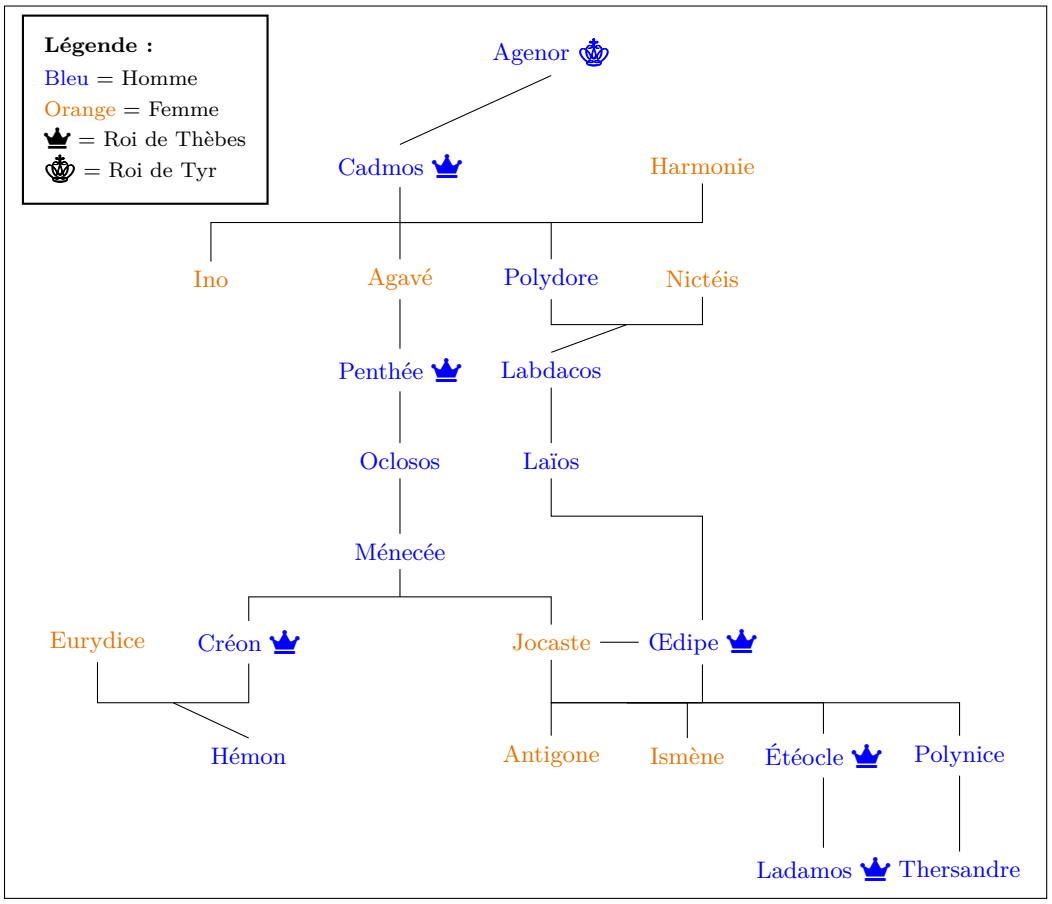

FIGURE 1 – La famille d'Oedipe

Où
Important : pour éviter d'être submergé sous l'ensemble des faits produits, vous pouvez demander à Clingo de visualiser seulement les atomes ayant un certain prédicat, grâce à la commande **#show** par exemple :

```
#show femme/1. % montre l'ensemble des faits sur le predicat unaire femme.
```

On doit indiquer l'arité du prédicat dans la commande car Clingo admet qu'on ait plusieurs prédicats de même nom (et d'arité différente). Vous pouvez ajouter plusieurs commandes **#show** à la fin de votre base de connaissances.

1. Définir les prédicats binaires **père** (X est père de Y), **mère** (X est mère de Y), **parent** (X est parent de Y). À chaque fois, vérifier que les faits déduits sont ceux attendus. Bien sûr, **parent** et **aEnfant** sont des synonymes.
2. Écrire une règle permettant de répondre à la question : qui sont les rois dont le père était déjà roi ? On n'a pas la notion de requête proprement dite, on la remplace par une règle dont le prédicat de tête fournit les réponses :

   ```
   answer(X) :- requete sur X et autres variables
   ```

   En affichant les atomes qui ont ce prédicat (**#show**), on obtient les réponses.
3. Écrire une règle permettant de répondre à la question : qui sont les rois dont le père était déjà roi du même lieu ?
4. Définir le prédicat **grand-parent** (X est grand-parent de Y), puis écrire une règle permettant de répondre à la question : qui sont les grands-parents d'Oedipe ?
5. Définir le prédicat **ancêtre** (X est ancêtre de Y) puis écrire une règle permettant de répondre à la question : qui sont les ancêtres d'Oedipe ? On considère que tout ascendant est un ancêtre.
6. Qui sont les personnages de sexe inconnu ? Vous aurez besoin ici de négation (not).
   On définit d'abord le prédicat **sexe_connu** :

   ```
   sexe_connu(X) :- femme(X).
   sexe_connu(X) :- homme(X).
   ```

   Ensuite on voudrait écrire :

   ```
   sexe_inconnu(X) :- not sexe_connu(X). % unsafe
   ```

   Cependant, Clingo n'admet que des règles dites sûres (safe) : toute variable apparaissant dans un littéral négatif doit aussi apparaitre dans un littéral positif du corps de la règle. Ici, X n'apparait pas dans un littéral positif du corps de la règle, Clingo ne sait donc pas quelles sont les valeurs possibles pour X.
   On va lui dire que X est un personnage :

   ```
   sexe_inconnu(X) :- personnage(X), not sexe_connu(X). % safe
   ```

   Tous nos personnages ont un sexe connu, mais vous pouvez commenter cette information pour certains personnages et vérifier que vos règles les retrouvent.

Solution.
Exercice solution

TP2 — Clingo avancé

Exercice 2.1 Coloration de graphes.
Nous nous intéressons maintenant à la modélisation de problèmes de satisfaction de contraintes, en commençant par la coloration de graphe. Un graphe est décrit grâce aux prédicats `node` et `edge`.
On rappelle les règles vues en cours permettant de générer toutes les assignations des noeuds à l'une des 3 couleurs red, green ou blue.

```
% generation de toutes les affectations
colored(X,red):- node(X),not colored(X,blue),not colored(X,green).
colored(X,blue):- node(X),not colored(X,red),not colored(X,green).
colored(X,green):- node(X),not colored(X,blue),not colored(X,red).
```

1. Ajouter la contrainte négative indiquant que deux sommets **adjacents** ne peuvent pas être colorés par la même couleur.
2. Appliquer le programme obtenu à la carte de l'Australie décrite par un graphe :

```
% 7 sommets (atomes unaires)
node(wa;nt;sa;q;nsw;v;t).
% 9 aretes (atomes binaires)
edge(sa,wa;sa,nt;sa,q;sa,nsw;sa,v;wa,nt;nt,q;q,nsw;nsw,v).
```

Vérifier que vous obtenez bien 18 solutions.

Solution.

1. On commence par ajouter la contrainte négative :

```
% deux sommets adjacents ne peuvent avoir la meme couleur
:- edge(X,Y), colored(X,Z), colored(Y,Z).
```

2. En appliquant le programme à la carte de l'Australie, on obtient bien 18 solutions :

```
% generation de toutes les affectations
colored(X,red):- node(X), not colored(X,blue), not colored(X,green).
colored(X,blue):- node(X), not colored(X,red), not colored(X,green).
colored(X,green):- node(X), not colored(X,blue), not colored(X,red).

% deux sommets adjacents ne peuvent avoir la meme couleur
:- edge(X,Y), colored(X,Z), colored(Y,Z).

% 7 sommets (atomes unaires)
node(wa;nt;sa;q;nsw;v;t).
% 9 aretes (atomes binaires)
edge(sa,wa;sa,nt;sa,q;sa,nsw;sa,v;wa,nt;nt,q;q,nsw;nsw,v).
```

On peut visualiser chacune de ces solutions en utilisant l'option **enumerate all** de Clingo.

Exercice 2.2 Configuration automobile.
On considère le problème de configuration suivant (cf. TD problèmes de satisfaction de contraintes). On rappelle ci-dessous l'énoncé global, mais dans les questions qui suivent on va résoudre ce problème étape par étape.
Une firme automobile élabore un nouveau modèle de voiture fabriquée dans toute l'Europe :

- les portières et le capot sont fabriqués à Lille où l'on ne dispose que de peinture rouge, jaune et noire ;
- la carrosserie est faite à Hambourg où l'on a de la peinture blanche, jaune, rouge et noire ;

- les pare-chocs, réalisés à Palerme, sont toujours blancs ;
- la bâche du toit ouvrant, faite à Madrid, ne peut être que blanche, jaune ou rouge ;
- les enjoliveurs sont fabriqués à Athènes où l'on a de la peinture rouge et jaune.

Le constructeur de la voiture a les exigences suivantes :

- la carrosserie, les portières et le capot sont de la même couleur ;
- les enjoliveurs, les pare-chocs et la bâche du toit ouvrant doivent être (strictement) plus clairs que la carrosserie (on considère que jaune est plus clair que rouge ; blanc et noir étant les deux extrêmes).

On souhaite déterminer l'ensemble des configurations possibles pour ce modèle. On se donne le prédicat unaire **objet** et les faits suivants pour énumérer les composants de la voiture :

```
objet(portiere;capot;carrosserie;parechoc;bache;enjoliveur)
```

On considère les couleurs blanc, jaune, rouge et noir, qu'on représente par les constantes b, j, r, n.

1. En admettant que chacun des 6 composants puisse être peint avec chacune des 4 couleurs, combien y a-t-il de configurations différentes possibles ?
   Écrire les règles qui permettent de générer toutes ces configurations (on utilisera un prédicat binaire **aCouleur**). Vérifier que vous obtenez bien le nombre de configurations voulu.
2. Représenter la condition « carrosserie, portières et capot sont de la même couleur » sous la forme de contraintes négatives. Combien de modèles (configurations) obtenez-vous maintenant ?
3. On s'intéresse maintenant à la notion de « plus clair ». On utilise le prédicat binaire plusClair(X, Y) pour dire que la couleur X est plus claire que la couleur Y. Ajoutez des faits, et peut-être des règles, pour définir cette notion sur les 4 couleurs.
4. Représenter la condition « enjoliveur, parechoc et bâche sont plus clairs que la carrosserie » sous forme de contraintes négatives. Combien de modèles obtenez-vous ?
5. Gérer maintenant les domaines sous forme de contraintes négatives qui interdisent certaines couleurs selon les composants d'une voiture.

Vous devriez obtenir 8 solutions au problème.

Solution.

1. Chacun des 6 composants pouvant avoir une de 4 couleurs, on a $4^6 = 4096$ configurations possibles. Le programme suivant génère toutes ces configurations :

```
% generation de toutes les affectations
aCouleur(X,b):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,j):- objet(X),not aCouleur(X,b),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,r):- objet(X),not aCouleur(X,j),not aCouleur(X,b),not aCouleur(X,n).
aCouleur(X,n):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,b).
```

2. On ajoute la contrainte négative suivante pour représenter la condition « carrosserie, portières et capot sont de la même couleur » :

```
% C1
:- aCouleur(carrosserie,U), aCouleur(portiere,V), U!=V.
:- aCouleur(carrosserie,U), aCouleur(capot,V), U!=V.
:- aCouleur(portiere,U), aCouleur(capot,V), U!=V.
```

   On obtient alors 256 configurations.

3. On ajoute les faits suivants pour définir la notion de « plus clair » :

```
plusClair(b,j). plusClair(b,r). plusClair(b, n).
plusClair(j,r). plusClair(j,n).
```

```
plusClair(r,n).
```

4. On ajoute la contrainte négative suivante pour représenter la condition « enjoliveur, parechoc et bâche sont plus clairs que la carrosserie » :

```
% C2
:- aCouleur(enjoliveur, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(parechoc, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(bache, U), aCouleur(carrosserie, V), not plusClair(U,V).
```

   On obtient alors 36 configurations.

5. On ajoute les contraintes négatives suivantes pour gérer les domaines :

```
% C3
% Lille
:- aCouleur(portiere, b). :- aCouleur(capot, b).

% Hambourg, aucune contrainte

% Palerme
:- aCouleur(parechoc, j). :- aCouleur(parechoc, r). :- aCouleur(parechoc,n).

% Madrid
:- aCouleur(bache, n).

% Athenes
:- aCouleur(enjoliveur,b). :- aCouleur(enjoliveur, n).
```

   On obtient alors 8 configurations, comme attendu.

Le programme complet est le suivant :

```
% faits
objet(portiere; capot; carrosserie; parechoc; bache; enjoliveur).
plusClair(b,j). plusClair(b,r). plusClair(b, n).
plusClair(j,r). plusClair(j,n).
plusClair(r,n).


% generation de toutes les affectations
aCouleur(X,b):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,j):- objet(X),not aCouleur(X,b),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,r):- objet(X),not aCouleur(X,j),not aCouleur(X,b),not aCouleur(X,n).
aCouleur(X,n):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,b).

% regles

% C1
:- aCouleur(carrosserie,U), aCouleur(portiere,V), U!=V.
:- aCouleur(carrosserie,U), aCouleur(capot,V), U!=V.
:- aCouleur(portiere,U), aCouleur(capot,V), U!=V.

% C2
:- aCouleur(enjoliveur, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(parechoc, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(bache, U), aCouleur(carrosserie, V), not plusClair(U,V).

% C3
% Lille
:- aCouleur(portiere, b). :- aCouleur(capot, b).
```

```
% Hambourg, aucune contrainte

% Palerme
:- aCouleur(parechoc, j). :- aCouleur(parechoc, r). :- aCouleur(parechoc,n).

% Madrid
:- aCouleur(bache, n).

% Athenes
:- aCouleur(enjoliveur,b). :- aCouleur(enjoliveur, n).

#show objet/1.
```

TP3 — Controle continu

Partie 1. La famille d'Oedipe (oedipe.txt)

Reprendre la base de faits d'Oedipe (le fichier `oedipe-family-facts.lp` qui vous a été fourni au TP1). Les prédicats de la base de faits qui nous intéressent sont :

- homme,
- femme,
- aEnfant.
- On n'utilisera pas le prédicat roi.

Quand on vous demande de définir un prédicat, vous pouvez définir des prédicats intermédiaires. Dans le fichier texte, donnez **toutes les règles utilisées** pour répondre aux questions.

Exercice 3.1.
Définir le prédicat unaire sansEnfant(X) qui est vrai si X n'a pas d'enfant connu. Combien de faits de prédicat sansEnfant(X) obtenez-vous ?

Solution.
On rappelle la structure de la famille d'Oedipe :

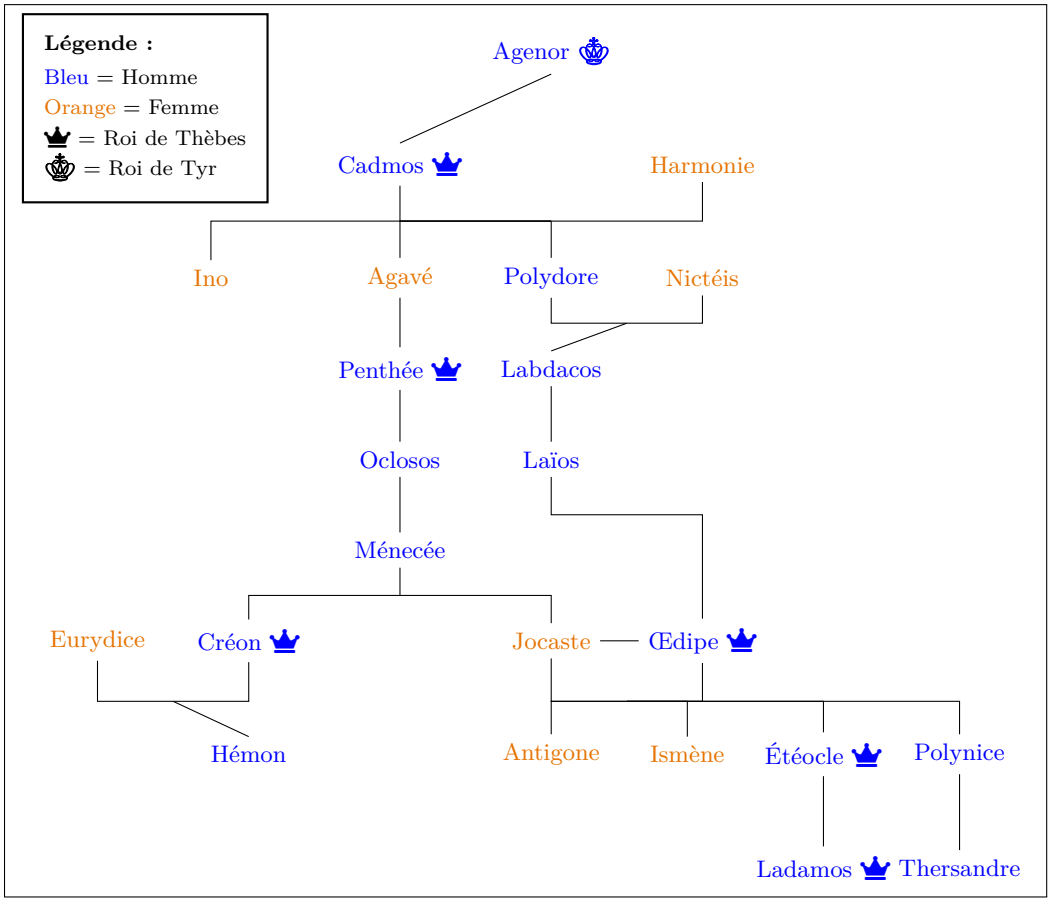

FIGURE 2 – La famille d'Oedipe

On définit le prédicat sansEnfant(X) comme suit :

```
% Question 1
parent(X) :- personnage(X), personnage(Y), aEnfant(X,Y), X != Y.
sansEnfant(X) :- personnage(X), not parent(X).
% #show parent/1.
#show sansEnfant/1.
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
sansEnfant(ino) sansEnfant(hemon) sansEnfant(antigone)
sansEnfant(ismene) sansEnfant(thersandre) sansEnfant(ladamos)
SATISFIABLE
```

On obtient 6 prédicats sansEnfant(X) qu'on peut facilement vérifier sur la figure 2.

Exercice 3.2.
Définir le prédicat unaire aAuMoinsDeuxEnfants(X) qui est vrai si X a au moins deux enfants. Combien de faits de prédicat aAuMoinsDeuxEnfants(X) obtenez-vous ?

Solution.
On définit le prédicat aAuMoinsDeuxEnfants(X) comme suit :

```
% Question 2
aAuMoinsDeuxEnfants(X) :- personnage(A), personnage(B), personnage(X),
A != B, A != X, B != X, aEnfant(X, A), aEnfant(X, B).
#show aAuMoinsDeuxEnfants/1.
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
aAuMoinsDeuxEnfants(cadmos) aAuMoinsDeuxEnfants(harmonie)
aAuMoinsDeuxEnfants(menecee) aAuMoinsDeuxEnfants(jocaste) aAuMoinsDeuxEnfants(oedipe)
SATISFIABLE
```

On obtient 5 prédicats aAuMoinsDeuxEnfants(X) qu'on peut facilement vérifier sur la figure 2.

Exercice 3.3.
Définir le prédicat unaire femmeEnfantUnique(X) qui est vrai si X est une femme ayant un unique enfant. Combien de faits de prédicat femmeEnfantUnique(X) obtenez-vous ?

Solution.
On définit le prédicat femmeEnfantUnique(X) comme suit :

```
% Question 3
femmeEnfantUnique(X) :- personnage(X), not sansEnfant(X),
not aAuMoinsDeuxEnfants(X), femme(X).
#show femmeEnfantUnique/1.
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
femmeEnfantUnique(agave) femmeEnfantUnique(nicteis) femmeEnfantUnique(eurydice)
SATISFIABLE
```

On obtient 3 prédicats femmeEnfantUnique(X) qu'on peut vérifier sur la figure 2.

Exercice 3.4.
Définir les choses suivantes :

- le prédicat binaire ancetre(X, Y) qui est vrai si X est un ancêtre de Y. On suppose ici que tout ascendant (y compris un parent) est un ancêtre et que personne n'est son propre ancêtre.
- le prédicat ternaire ancetreCommun(Z, X, Y) qui est vrai si Z est un ancêtre commun de X et Y (avec $X \neq Y$).
- le prédicat answer qui donne la réponse à la question « Qui sont les plus lointains ancêtre communs à **Antigone** et **Laios** ? » où un plus loin ancêtre commun est un ancêtre commun dont on ne connait pas de parent.

Solution.

- On définit le prédicat ancetre(X, Y) comme suit :

```
% Question 4 - 1
ancetre(X,Y) :- aEnfant(X,Y).
ancetre(X,Y) :- ancetre(X,Z), ancetre(Z,Y).
#show ancetre/2.
```

- On définit le prédicat ancetreCommun(Z, X, Y) comme suit :

```
% Question 4 - 2
ancetreCommun(Z,X,Y) :- ancetre(Z,X), ancetre(Z,Y), X != Y.
#show ancetreCommun/3.
```

- On définit le prédicat answer comme suit :

```
% Question 4 - 3
aUnParent(X) :- personnage(Y), aEnfant(Y,X), X != Y.
answer(Z) :- personnage(Z), personnage(U),
ancetreCommun(Z,antigone,laios), not aUnParent(Z).
#show answer/1.
```

En lançant Clingo, on obtient la sortie suivante :

```
Answer: 1
answer(nicteis) answer(harmonie) answer(agenor)
SATISFIABLE
```

La réponse est donc que les plus lointains ancêtres communs à Antigone et Laios sont : Nicteis, Harmonie et Agenor.

Le code complet est donné en annexe A.

Partie 2. Configuration automobile (config.txt)

On reprend le problème de configuration automobile du TP2 dont on rappelle l'énoncé ci-dessous. Une firme automobile élabore un nouveau modèle de voiture fabriquée dans toute l'Europe :

- les portières et le capot sont fabriqués à Lille où l'on ne dispose que de peinture rouge, jaune et noire ;
- la carrosserie est faite à Hambourg où l'on a de la peinture blanche, jaune, rouge et noire ;
- les pare-chocs, réalisés à Palerme, sont toujours blancs ;
- la bâche du toit ouvrant, faite à Madrid, ne peut être que blanche, jaune ou rouge ;
- les enjoliveurs sont fabriqués à Athènes où l'on a de la peinture rouge et jaune.

Le constructeur de la voiture a les exigences suivantes :

- la carrosserie, les portières et le capot sont de la même couleur ;
- les enjoliveurs, les pare-chocs et la bâche du toit ouvrant doivent être (strictement) plus clairs que la carrosserie (on considère que jaune est plus clair que rouge ; blanc et noir étant les deux extrêmes).

On souhaite déterminer l'ensemble des configurations possibles pour ce modèle. On se donne le prédicat unaire **objet** et les faits suivants pour énumérer les composants de la voiture :

```
objet(portiere;capot;carrosserie;parechoc;bache;enjoliveur)
```

On considère les couleurs blanc, jaune, rouge et noir, qu'on représente par les constantes b, j, r, n.

Exercice 3.5.
Rendre votre programme ASP qui répond au problème de configuration automobile complet en y ajoutant les contraintes correspondant aux exigences du constructeur.

Solution.
Voir annexe B pour le code complet.

Exercice 3.6.
Pour gérer la notion de « couleur plus claire » utilisant le prédicat binaire plusClair(X, Y), avez-vous introduit seulement des faits dans votre programme ou également une règle (dans ce cas, donnez cette règle) ?
Dans chaque modèle stable, combien obtenez-vous de faits ayant le prédicat plusClair ?

Solution.
On a introduit seulement des faits pour le prédicat plusClair(X, Y) :

```
plusClair(b,j). plusClair(b,r). plusClair(b, n).
plusClair(j,r). plusClair(j,n).
plusClair(r,n).
```

On obtient 6 faits ayant le prédicat plusClair dans chaque modèle stable.

Exercice 3.7.
Par quelles contraintes négatives avez-vous représenté la condition « l'enjoliveur, le pare-chocs et la bâche sont plus clairs que la carrosserie » ?

Solution.
On a utilisé les contraintes négatives suivantes :

```
% C2
:- aCouleur(enjoliveur, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(parechoc, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(bache, U), aCouleur(carrosserie, V), not plusClair(U,V).
```

Exercice 3.8.
Vous semble-t-il possible d'écrire la condition de la question précédente sous forme de règles positives ? Expliquez (donnez les règles positives qui remplaçent les contraintes négatives si vous pensez que c'est possible et sinon expliquez pourquoi ce n'est pas possible).

Solution.
En utilisant des conditions négatives on retire des configurations interdites à l'ensemble des configurations possibles, qu'on a généré au début. Si on ne change pas cette partie au début il restera toujours des configurations interdites.
On peut utiliser des règles positives pour générer uniquement des configurations valides, mais dans ce cas il faut réécrire toute la partie de génération des configurations possibles au début.

Partie 3. Modélisation d'un autre problème de satisfaction de contraintes (table.txt)

On considère le problème suivant :

Il y a 6 sièges autour d'une table ronde et 6 invités à placer sur ces sièges. Le but est de trouver toutes les façons d'affecter un siège à un invité sachant qu'il faut respecter certaines contraintes liées aux relations de sympathie ou antipathie entre les invités.
Voici un squelette de programme ASP dans lequel on utilise deux prédicats : guest(X) pour dire que X est un invité et at(X, Y) pour dire que X est placé au siège Y.
On utilise aussi la règle du choix comme raccourci pour éviter d'avoir 6 règles disant que tout invité a un siège parmi les 6 (prenez cette règle telle quelle, vous n'aurez pas à la modifier) :

```
% predicat unaire guest, les 6 invites sont:
% a, b, c, d, e, f
guest(a;b;c;d;e;f).

% tout invite a exactement une place prise entre 1 et 6
1 { at (X,1..6)} 1 :- guest(X).
#show at/2
```

Exercice 3.9.
Ajouter une contrainte exprimant qu'une place ne peut pas recevoir deux invités.

Solution.
On ajoute la contrainte suivante :

```
% Question 1
:- at(X,A), at(Y,A), X != Y.
```

Exercice 3.10.
Par quelle formule définir le nombre de modèles stables à cette étape ? (Vous pouvez aussi donner le nombre de modèles stables mais l'important est de donner la formule qui permet de le calculer.)

Solution.
Le nombre de modèles stables est donné par la formule 6! soit 720. Elle vient du fait que le premier invité pour choisir n'importe laquelle des 6 places de départ, le suivant n'a plus que 5 places possibles, le suivant 4, etc.

Exercice 3.11.
Définir la notion de « côte à côte » grâce au prédicat binaire nextTo(X, Y) qui est vrai si X et Y ont des places adjacentes autour de la table, qui, rappelons-le, est ronde (ainsi les places 1 et 6 sont adjacentes).
Conseil : faites simple pour commencer et vous pourrez faire plus élégant si vous en avez le temps à la fin de l'exercice.

Solution.
On définit le prédicat nextTo(X, Y) comme suit :

```
% Question 3
aGauche(X,Y) :- at(X,A), at(Y,B), A == B + 1.
aDroite(X,Y) :- at(X,A), at(Y,B), A == B - 1.
cas1(X,Y) :- at(X,A), at(Y,B), A == 1, B == 6.
cas2(X,Y) :- at(X,A), at(Y,B), A == 6, B == 1.
nextTo(X,Y) :- aGauche(X,Y).
nextTo(X,Y) :- aDroite(X,Y).
nextTo(X,Y) :- cas1(X,Y).
nextTo(X,Y) :- cas2(X,Y).
% #show nextTo/2.
```

On gère initialement les cas simples de gauche et droite, puis on ajoute les cas particuliers des extrémités 1 et 6.

Exercice 3.12.
On gère maintenant deux contraintes de placement :

1. Certains invités s'apprécient particulièrement (prédicat binaire like(X, Y)) et on veut vraiment les placer côte à côte.
2. D'autres invités ne s'apprécient pas du tout (prédicat binaire dislike(X, Y)) et on veut absolument éviter de les placer côte à côte.

On a ainsi les faits suivants :

```
like(a,b). like(c,d).
dislike(b,c). dislike(a,c).
```

Ajouter à votre programme les contraintes de placement correspondantes. Combien de solutions obtenez-vous ?

Solution.
On ajoute les contraintes suivantes :

```
:- like(X,Y), not nextTo(X,Y).
:- like(Y,X), not nextTo(X,Y).

:- dislike(X,Y), nextTo(X,Y).
:- dislike(Y,X), nextTo(X,Y).
```

En lançant Clingo, on obtient 96 solutions respectant les contraintes de placement.

Le code complet est donné en annexe C.

Partie 4. Configuration auomobile 2 (config2.txt)

On reprend le problème de configuration automobile. Dans le TP, on vous demandait de gérer les domaines sous forme de contraintes négatives qui interdisent certaines couleurs selon les composants d'une voiture. On vous demande maintenant une deuxième version du programme où les domaines sont donnés sous forme de faits en utilisant un prédicat binaire aColPossible(X, Y) qui dit que X a pour couleur possible Y. Par exemple, pour les portières on aurait :

```
aColPossible(portiere, j). aColPossible(portiere, r). aColPossible(portiere, n).
```

Dans un fichier texte nommé `config2.txt`, donner une autre version des règles qui génèrent toutes les affectations de façon à pouvoir supprimer les contraintes négatives qui interdisaient aux composants d'avoir des couleurs hors de leur domaine. Obtenez-vous un résultat satisfaisant ?

Solution.
Voir annexe D pour le code complet.
Le résultat est satisfaisant, on obtient le même nombre de configurations possibles que dans la première version du programme à condition encore une fois de modifier la partie de génération des configurations possibles.

A Code source complet pour la famille d'Oedipe

Le code source complet utilisé pour les exercices sur la famille d'Oedipe est donné ci-dessous.

```
% oedipe-family-factbase.lp

% les caracteres accentues ne sont pas admis par clingo
% ou alors il faut les mettre entre guillemets

% BASE DE FAITS

% personnage est un predicat unaire
% on utilise une ecriture raccourcie permise par Clingo:
% p(a;b). est un raccourci pour p(a). p(b).
% Ne pas confondre p(a;b) et p(a,b)
personnage(agenor;cadmos;harmonie;ino;agave;polydore;labdacos;nicteis;penthe;oclasos).
personnage(menecee;jocaste;creon;hemon;eurydice;laios;oedipe).
personnage(antigone;ismene;eteocle;polynice;thersandre;ladamos).

% sexe des personnages
homme(agenor). % agenor
homme(cadmos).
femme(harmonie).
femme(ino).
femme(agave).   % agave
homme(polydore).
homme(labdacos).
femme(nicteis).   % nicteis
homme(penthe). % penthe
homme(oclasos).
homme(menecee). % menecee
femme(jocaste).
homme(creon).   % creon
homme(hemon). % hemon
femme(eurydice).
homme(laios).
homme(oedipe).
femme(antigone).
femme(ismene).   % ismene
homme(eteocle). % eteocle
homme(polynice).
homme(thersandre).
homme(ladamos).

% relations parent-enfant
aEnfant(agenor,cadmos).
aEnfant(cadmos,ino).
aEnfant(cadmos,agave).
aEnfant(cadmos,polydore).
aEnfant(harmonie,ino).
aEnfant(harmonie,agave).
aEnfant(harmonie,polydore).
aEnfant(polydore,labdacos).
aEnfant(nicteis,labdacos).
aEnfant(agave,penthe).
aEnfant(labdacos,laios).
aEnfant(penthe,oclasos).
aEnfant(oclasos,menecee).
aEnfant(menecee,jocaste).
aEnfant(menecee,creon).
aEnfant(creon,hemon).
```

```
aEnfant(eurydice,hemon).
aEnfant(laios,oedipe).
aEnfant(jocaste,oedipe).
aEnfant(jocaste,antigone).
aEnfant(jocaste,eteocle).
aEnfant(jocaste,ismene).
aEnfant(jocaste,polynice).
aEnfant(oedipe,antigone).
aEnfant(oedipe,eteocle).
aEnfant(oedipe,ismene).
aEnfant(oedipe,polynice).
aEnfant(polynice, thersandre).
aEnfant(eteocle,ladamos).

% ------------------------------------------------

% Question 1
parent(X) :- personnage(X), personnage(Y), aEnfant(X,Y), X != Y.
sansEnfant(X) :- personnage(X), not parent(X).
% #show parent/1.
% #show sansEnfant/1.

% Question 2
aAuMoinsDeuxEnfants(X) :- personnage(A), personnage(B), personnage(X),
A != B, A != X, B != X, aEnfant(X, A), aEnfant(X, B).
% #show aAuMoinsDeuxEnfants/1.


% Question 3
femmeEnfantUnique(X) :- personnage(X), not sansEnfant(X),
not aAuMoinsDeuxEnfants(X), femme(X).
% #show femmeEnfantUnique/1.

% Question 4
% Question 4 - 1
ancetre(X,Y) :- aEnfant(X,Y).
ancetre(X,Y) :- ancetre(X,Z), ancetre(Z,Y).
% #show ancetre/2.
% Question 4 - 2
ancetreCommun(Z,X,Y) :- ancetre(Z,X), ancetre(Z,Y), X != Y.
% #show ancetreCommun/3.
% Question 4 - 3
aUnParent(X) :- personnage(Y), aEnfant(Y,X), X != Y.
answer(Z) :- personnage(Z), personnage(U),
ancetreCommun(Z,antigone,laios), not aUnParent(Z).
% #show answer/1.
```

B Code source complet pour la configuration automobile

Le code source complet utilisé pour les exercices sur la configuration automobile est donné ci-dessous.

```
% faits
objet(portiere; capot; carrosserie; parechoc; bache; enjoliveur).
plusClair(b,j). plusClair(b,r). plusClair(b, n).
plusClair(j,r). plusClair(j,n).
plusClair(r,n).


% generation de toutes les affectations
```

```
aCouleur(X,b):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,j):- objet(X),not aCouleur(X,b),not aCouleur(X,r),not aCouleur(X,n).
aCouleur(X,r):- objet(X),not aCouleur(X,j),not aCouleur(X,b),not aCouleur(X,n).
aCouleur(X,n):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,b).

% regles

% C1
:- aCouleur(carrosserie,U), aCouleur(portiere,V), U!=V.
:- aCouleur(carrosserie,U), aCouleur(capot,V), U!=V.
:- aCouleur(portiere,U), aCouleur(capot,V), U!=V.

% C2
:- aCouleur(enjoliveur, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(parechoc, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(bache, U), aCouleur(carrosserie, V), not plusClair(U,V).

% C3
% Lille
:- aCouleur(portiere, b). :- aCouleur(capot, b).

% Hambourg, aucune contrainte

% Palerme
:- aCouleur(parechoc, j). :- aCouleur(parechoc, r). :- aCouleur(parechoc,n).

% Madrid
:- aCouleur(bache, n).

% Athenes
:- aCouleur(enjoliveur,b). :- aCouleur(enjoliveur, n).

#show objet/1.
```

C Code source complet pour le problème de satisfaction de contraintes

Le code source complet utilisé pour les exercices sur le problème de satisfaction de contraintes est donné ci-dessous.

```
% predicat unaire guest, les 6 invites sont:
% a, b, c, d, e, f
guest(a;b;c;d;e;f).

% tout invite a exactement une place prise entre 1 et 6
1 { at (X,1..6) } 1 :- guest(X).

% Question 1
:- at(X,A), at(Y,A), X != Y.

% Question 3
aGauche(X,Y) :- at(X,A), at(Y,B), A == B + 1.
aDroite(X,Y) :- at(X,A), at(Y,B), A == B - 1.
cas1(X,Y) :- at(X,A), at(Y,B), A == 1, B == 6.
cas2(X,Y) :- at(X,A), at(Y,B), A == 6, B == 1.
nextTo(X,Y) :- aGauche(X,Y).
nextTo(X,Y) :- aDroite(X,Y).
nextTo(X,Y) :- cas1(X,Y).
nextTo(X,Y) :- cas2(X,Y).
```

```
% #show nextTo/2.

% Question 4
like(a,b). like(c,d).
dislike(b,c). dislike(a,c).

:- like(X,Y), not nextTo(X,Y).
:- like(Y,X), not nextTo(X,Y).

:- dislike(X,Y), nextTo(X,Y).
:- dislike(Y,X), nextTo(X,Y).

#show at/2.
```

D Code source complet pour la configuration automobile 2

Le code source complet utilisé pour les exercices sur la configuration automobile 2 est donné ci-dessous.

```
% faits
objet(portiere; capot; carrosserie; parechoc; bache; enjoliveur).
plusClair(b,j). plusClair(b,r). plusClair(b, n).
plusClair(j,r). plusClair(j,n).
plusClair(r,n).


% generation de toutes les affectations
aCouleur(X,b):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,n),
aColPossible(X,b).
aCouleur(X,j):- objet(X),not aCouleur(X,b),not aCouleur(X,r),not aCouleur(X,n),
aColPossible(X,j).
aCouleur(X,r):- objet(X),not aCouleur(X,j),not aCouleur(X,b),not aCouleur(X,n),
aColPossible(X,r).
aCouleur(X,n):- objet(X),not aCouleur(X,j),not aCouleur(X,r),not aCouleur(X,b),
aColPossible(X,n).

% regles

% C1
:- aCouleur(carrosserie,U), aCouleur(portiere,V), U!=V.
:- aCouleur(carrosserie,U), aCouleur(capot,V), U!=V.
:- aCouleur(portiere,U), aCouleur(capot,V), U!=V.

% C2
:- aCouleur(enjoliveur, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(parechoc, U), aCouleur(carrosserie, V), not plusClair(U,V).
:- aCouleur(bache, U), aCouleur(carrosserie, V), not plusClair(U,V).

% C3 c'est ici qu'on modifie
% Lille
aColPossible(portiere, j). aColPossible(portiere, r). aColPossible(portiere, n).
aColPossible(capot, j). aColPossible(capot, r). aColPossible(capot, n).

% Hambourg, aucune contrainte
aColPossible(carrosserie, j).
aColPossible(carrosserie, r).
aColPossible(carrosserie, n).
aColPossible(carrosserie, b).

% Palerme
```

```
aColPossible(parechoc, b).

% Madrid
aColPossible(bache, b). aColPossible(bache, j). aColPossible(bache, r).

% Athenes
aColPossible(enjoliveur,j).
aColPossible(enjoliveur,r).

#show objet/1.
```