

Introduction à l'IA symbolique — TPs

Ivan Lejeune

9 décembre 2025

Table des matières

TP1 — Introduction à Clingo.	2
TP2 — Clingo avancé	7

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 prédictats unaires **animal**, **plante**, **carnivore**, **herbivore**, **omnivore** et **humain**, et le prédictat 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 \neq 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** » :

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

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(chvre). % (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(chvre)
```

```

herbivore(chvre) herbivore(chaperon) omnivore(chaperon) humain(chaperon)
mange(loup,chevre) mange(chaperon,chevre) mange(loup,chaperon)
mange(chaperon,chaperon) plante(chvre) 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(chvre)
herbivore(chvre) herbivore(chaperon) omnivore(chaperon) humain(chaperon)
mange(loup,chevre) mange(chaperon,chevre) mange(loup,chaperon) plante(chvre)
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(chvre) animal(chaperon)
herbivore(chvre) 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(chvre). % (a)
carnivore(loup). % (b)
humain(chaperon). % (c)

% règles
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 négative

```

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 :

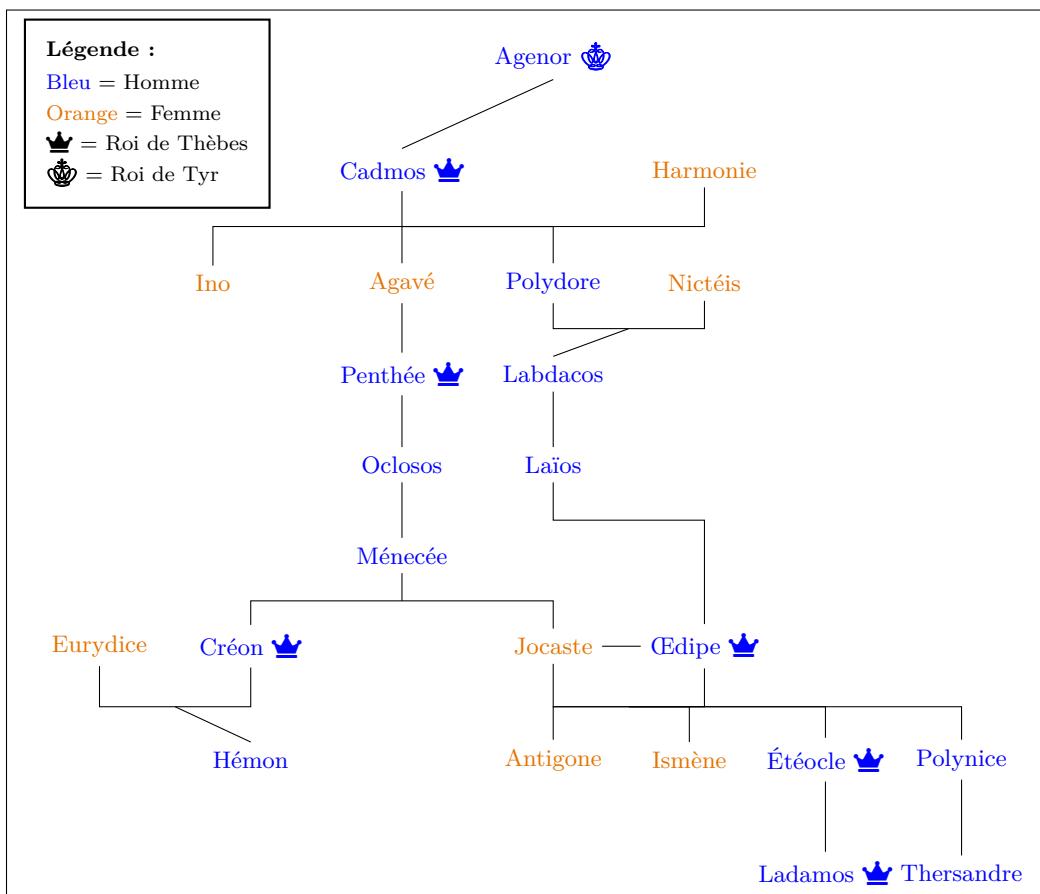


FIGURE 1 – La famille d’Oedipe

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 prédicat 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.

- Définir les prédictats 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.
- 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 prédictat de tête fournit les réponses :

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

En affichant les atomes qui ont ce prédictat (`#show`), on obtient les réponses.

- Écrire une règle permettant de répondre à la question : qui sont les rois dont le père était déjà roi du même lieu ?
- Définir le prédictat **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 ?
- Définir le prédictat **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 descendant est un ancêtre.
- Qui sont les personnages de sexe inconnu ? Vous aurez besoin ici de négation (`not`). On définit d'abord le prédictat **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 apparaître 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 prédictats `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.

```
% génération 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 arêtes (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 même 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 :

```
% génération 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 même couleur
:- edge(X,Y), colored(X,Z), colored(Y,Z).

% 7 sommets (atomes unaires)
node(wa;nt;sa;q;nsw;v;t).
% 9 arêtes (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 :

```
% génération 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).

% génération 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).

% règles

% 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.
```