Algorithmes & Raisonnement

2023-2024

BE1 : Programmation avec Prolog

1 Prise en main de Prolog (Vue en TD1)

Les séances de BE de ce module utilisent le langage **Prolog** (version swi-Prolog).
Avant de commencer le BE, vérifiez en section **??** en page **??** que vous avez bien installé le langage Prolog selon les plateformes Linux, Mac ou Windows.

On suppose ci-dessous que vous savez éditer et charger un programme Prolog.

1.1 Premiers Tests

Comme en TD1, un premier programme peut contenir :

```
mortel(X) :- homme(X).   % une règle : "∀X , X est mortel si X est un homme"
homme(socrates).         % un fait : "socrates est un homme".
homme(aristote).   % laisser en minuscule (Aristote avec 'A' = une variable)
homme(man).
```

NB : ce programme contient une **règle** et 3 **faits**.

- *Une* **règle** *est de la forme : partie gauche :- partie droite.*
- *Intuitivement, pour démontrer la partie gauche d'une règle, il faudra démontrer sa partie droite.*
- *Un* **fait** *est avéré (c'est comme une règle dont la partie droite = true.)*
- *Une variable commence par une lettre MAJUSCULE (ou par '_').*

Maintenant, on va poser des questions (ne pas taper "?-" qui est l'invite de Prolog):

```
?- mortel(X).        % Qui est mortel ? (∃X telle que mortel(X)=vrai ?)
→ X=socrates ?       % taper ";"  devant '?' pour avoir la réponse suivante.
→ X=aristote  ?
→ X=man
→ true
```

Remarques :

1- Sous Prolog, utiliser les **4 flèches** du pavé à droite pour rappeler les requêtes précédentes.

Par exemple, à cet endroit, la flèche haute rappelle "mortel(X)." tapée en dernier.

2- Pour obtenir les réponses à une requête :

- si la requête n'a pas de réponse, on obtient "no",
- si la requête a une seule réponse, on obtient la réponse suivi de "true"
- si la requête a plusieurs réponse, la première est donnée et Prolog attend notre réaction :

- on appuie sur le "point virgule" pour la réponse suivante
- on appuie sur "retour charriot" pour stopper les autres réponses.

- Controle-C + "retour chariot" arrête les choses brutalement (abandon)

3- Un autre intérêt d'avoir les réponses une-par-une est de pouvoir arrêter facilement une résolution en cas de *boucle infinie*

1.2 Charger son programme

Sous l'éditeur, produire le programme 'menu.pl' suivant (utiliser couper-coller).

/ Programme menu.pl : MENU au restaurant */*

entree(salade).
entree(oeufs).
entree(pate).
entree(melon).

plat(poisson).
plat(poulet).
plat(viande).

dessert(glace).
dessert(pomme).
dessert(raisin). *dessert(gateau).* *dessert(fromage).*

Comment / où placer vos programme Prolog ?

1. Vous pouvez placer vos fichiers Prolog (d'extension ".pl") dans le répertoire où vous avez installé Prolog. L'inconvénient est de mélanger vos fichiers avec ceux du Prolog !

2. Vous pouvez décider d'un répertoire (disons "/xx/yy/BE") et y placer vos programmes Prolog. Admettons que vous avez dans ce répertoire le programme "ex1.pl" (extension ".pl" obligatoire).
   Pour le charger, vous pourrez taper (sous Prolog):
   consult('/xx/yy/BE/ex1.pl'). ou ['/xx/yy/BE/ex1.pl'].
   Sous Windows : consult('C : \\xx\\yy\\BE\\ex1.pl'). ou ['C : \\xx\\yy\\BE\\ex1.pl'].

3. Vous pouvez changer de répertoire par la requête "**cd('nom répertoire').**"
   ☞ Sous swi-prolog, en plus simple : $cd('C : /xx/yy/BE')$.
   ☞ Sous Prolog, : $cd('C : \xx\yy\BE')$.
   Une fois placé dans dans le répertoire de vos programmes, vous aurez juste besoin de donner le nom du programme. Par exemple, *consult('ex1.pl').* ou *['ex1.pl'].*

4. Aussi, vous pouvez vous placer dans le répertoire de vos programme ("/xx/yy/BE") et de lancer Prolog. Pour cela, il faut placer le chemin d'accès à Prolog dans la variable PATH.
   Demandez à votre enseignant comment faire. La Doc de Prolog l'explique également.

... On reprend ... Charger ce programme par :

?- ['menu.pl'].
Ou par :
?- consult('menu.pl').

Vous pouvez aussi écrire "consult(" puis glisser le fichier "menu.pl" après la parenthèse puis ajouter ").".

➙ Le chargement d'un programme Prolog se fait par l'une des commandes Prolog suivantes :

consult(menu). %pour un nom de fichier simple, sinon, entourez de '
consult('menu.pl').
consult('/home/moi/TPs/menu.pl').
consult('C:/Documentsand Setting/moi/TPs/menu.pl').

ou de manière équivalente, en tapant l'une des requêtes suivantes (n'oubliez pas le '.' final) :

[menu].
['menu.pl'].
['/home/moi/TPs/menu.pl'].

L'extension par défaut des programmes Prolog est ".pl". Si vous utilisez une autre convention ou un nom de fichier composé (par exemple, "menu-repas.txt", quoique pas une bonne idée !), chargez votre programme par son nom complet ['menu-repas.txt'].

- Le caractère '**%**' marque le début d'un *commentaire* sur la même ligne.

On peut aussi insérer des commentaires sur plusieurs lignes par */* .....sur plusieurs lignes ..... */*

- Au besoin, sous Prolog, on peut visualiser les prédicats chargés par (intéressant si l'on a peu de prédicats dans la base) :

?- listing(entree). *% liste le prédicat 'entree'*

Un prédicat = un paquet de règles / faits dont le nom commence par le même symbole de prédicat et le même nombre de paramètres. Par exemple, le prédicat" entree (.)".

La convention Prolog est de noter cela " entree/1 " : nom du prédicat = " entree " et il a un seul paramètre. Un autre exemple était " homme/1 " ci-dessus...

- Si votre programme contient des erreurs ou si vous devez changer son contenu, faites les modifications sous l'éditeur et recharger le programmer à l'aide de la commande Prolog "*consult*" ou par *[...]* (comme ci-dessus).

Poser des questions :

?- entree(X).

→ *.... les réponses.*

2 Exercices en séance (menu)

I) A ce stade, nous avons chargé le programme suivant sous Prolog. Si vous avez eu des messages d'erreurs, il est impératif de les corriger sous l'éditeur puis de recharger le code.

/ Programme menu.pl : MENU au restaurant */*

entree(salade).
entree(oeufs).
entree(pate).
entree(melon).

plat(poisson).
plat(poulet).
plat(viande).

dessert(glace).
dessert(pomme).
dessert(raisin). *dessert(gateau).* *dessert(fromage).*

II) Maintenant, composer sous Prolog les questions suivantes. L'enseignant vous aidera à comprendre les réponses et vous donnera des compléments (voir remarques en fin de ce document).

1. *Quels sont les entrées, (puis) les plats, (puis) les desserts ? (poser 3 questions).*

   Solution (les réponses fournies par Prolog sont les faits donnés dans " menu.pl "):

   ?- entree(X).
   ?- plat(X).
   ?- dessert(X).

1. *Y a t-il du fromage en entrée ?*

   Quelle différence entre ces 3 formulations ? :

   entree(fromage). *entree(X), X=fromage.* *X = fromage, entree(X).*

1. *Est-ce que le gâteau est un plat?*

2. *Sachant qu'un menu est composé d'une entrée, un plat et un dessert, est-ce que le menu propose des frites ?*

   ➙ *Selon votre requête, la réponse peut être globale (Oui/non) ou bien on peut savoir si les frites sont servies en entrée et/ou comme plat et/ou au dessert.*

3. *Quels sont tous les repas que l'on peut composer dans ce menu ? (remarquer l'ordre des réponses).*

4. *Peut-on avoir une entrée et un dessert identiques.*

5. *Quel est le sens de la requête suivante :*

   entree(E1), entree(E2), E1 \= E2, plat(P). ' \=' veut dire : deux valeurs différentes.

Remarque sur '\=' :

La différence ('\=') est codée sous Prolog par la négation de l'égalité '='.

Si X et Y ont chacune une valeur, le résultat de ce test dépendra des valeurs de X et de Y.

☞ N'utilisez pas '\=' si X et/ou Y sont des variables. Testez :

?- toto \= titi. % oui, 2 constantes différentes

?- X \= Y. % non car X=Y est vrai !

?- X \= melon. % non car X=melon est vrai !

?- X= banane, X \= melon. % oui, X a une valeur (on compare *melon* et *banane*).

Donc, si on demande (question ci-dessus) :

?- E1 \= E2, entree(E1), entree(E2), plat(P).

➙ Cette fois, la réponse est Non car dès le départ, *E1 \= E2* échoue (deux variables ne sont pas différentes) et fera échouer toute la requête.

III)- Modifier le programme sous l'éditeur et ajouter la règle **repas(Entree, Plat, Dessert)** permettant de constituer un repas quelconque par ce menu (on a vu un exemple de *règle* avec *mortel(X) :- ....*). Ensuite, recharger votre programme puis poser les questions suivantes (à l'aide de **repas/3**) :

1. *Quels sont* **tous** *les* **repas***?*
2. *Quels sont les repas dont le* **plat** *(2e élément d'un menu) est poulet ?*
3. *Quels sont les repas où le* **dessert** *(3e élément) est une glace ?*
4. *Peut-on composer un repas dont* **l'entrée** *et le* **dessert** *sont identiques ?*
5. *Y a t-il des repas avec du poisson MAIS sans gâteau . Voir la note sur la négation ci-dessous.*

IV) Avec l'aide de l'enseignant, utiliser l'outil de trace de Prolog.
Voir en annexe-A les explications de ce mode.

La commande pour lancer la trace : **trace** (ou **debug** qui a besoin de points " spy ").

Pour désactiver la trace : **notrace** (ou **nodebug**).

Étudier le comportement du moteur Prolog sur la question 1 ci-dessus pour constater le mode de fonctionnement du moteur Prolog. Cela vous sera très utile pour la suite.

V) Modifier le programme et ajouter les valeurs calorifiques suivantes dans les faits (Lire ci-dessous pour les solutions possibles) :

salade : 100 *oeufs : 200* *pate : 300*

poisson : 400 *poulet : 500* *viande : 800*

glace : 300 *fromage : 320* *gateau : 400*

fruits : 250

Ajouter ensuite la règle **repas(Entree, Plat, Dessert, Valeur)** où *Valeur* est la somme des valeurs calorifiques du repas composé.

Indication pour les valeurs calorifiques:
une manière simple d'ajouter ces valeurs est de transformer, par exemple *entree(salade)* en *entree(salade, 100).*

Une question du type *entree(Ent, Val).*
donnera pour chaque entrée *Ent* sa valeur calorifique dans *Val.*

Une autre manière d'inclure ces valeurs est de laisser la base en l'état mais d'ajouter des faits nouveaux pour ces valeurs. Par exemple, on ajoutera (idem pour les autres ingrédients) :

calorifique(salade, 100).

Dans ce dernier cas, l'accès à la valeur calorifique d'une entrée se fait par

entree(E), calorifique(E, Val).

L'intérêt de cette deuxième méthode et de ne pas toucher à la BD précédente et donc aux programmes qui fonctionnent avec ce format sur entree/1.

Voir le point sur l'arithmétique en Prolog en page suivante.

Suite des exercices en séance

... Quelque soit la solution choisie pour représenter les calories, formuler les requêtes suivantes :

1. *Constituer un/des repas dont la valeur calorifique est inférieure a 800 (voir ci-dessous pour l'arithmétique).*
2. *Constituer un/des repas dont la valeur cal. est égale a une constante c.*
3. *Écrire la règle 'repas_leger' qui propose les repas dont la valeur cal. est entre deux constantes (e.g. entre 800 et 1000 cal.).*
4. *Utiliser vos règles pour composer un repas sans viande (poulet et poisson permis) dont 800 < valeur cal. < 900.*
5. *Supposons que vous n'aimez pas le dessert et que vous pouvez prendre deux entrées* **différentes**. *Composez un repas dont la val. Cal. est inférieur à 800 cal.*

Remarque : le calcul du repas d'une valeur calorifique minimum sera traité plus tard.

2.1 Point sur l'arithmétique

Arithmétique en Prolog standard :

Cette section donne quelques éléments sur l'arithmétique en Prolog Standard.

Pour faire de l'arithmétique simple (sans les contraintes) en Prolog, on utilise les opérateurs **:**
'is' '-' '+' '*', >, <, =<, >=, ... (voir la doc. Prolog)

Par exemple :

?- A=10, X is A + 5 - 2.

→ A=10 et X=13

?- entree(E), calorifique(E, Val), Val < 50 .

→ Succès si la valeur calorifique de E est < 50.

N.B. : le test "Val < 50" ici doit être placé là où Val possède une valeur. On verra par la suite comment les transformer en contraintes et les placer où l'on veut.

Arithmétique à l'aide des contraintes :

☞ Sous swi-Prolog, exécutez la requête use_module(library(clpfd)). pour avoir accès aux contraintes du domaine fini.

Les opérateurs ci-dessus s'appliquent seulement aux nombres. Enlever le '#' pour les appliquer aux autres termes (cf. ci-dessus).

Exemples d'arithmétique (les # permettent de distinguer les opérateurs arithmétiques de ceux de Prolog Standard) :

A + B #=10, A=1. % n'oubliez pas le '#' : c'est une équation sur les entiers positifs/nuls

```
% La résolution de cette équation donne :
A = 1,
B = 9.
```

*X + 2 * Y #= Z + 2, X=1.*

```
% Pour celle-ci, on obtient simplement une simplification :
X = 1,
2*Y #= Z+1.   % "#=" est l'équalité équationnelle : 2Y=Z+1
```

A #> 5 - Y. % n'oubliez pas le '#' : c'est une inéquation sur les N

```
% Ici, on obtient :
_A in 6..sup,    % La variable temporaire _A est un entier dans 6.. infini
A+Y#=_A.         % et A+Y = La variable temporaire _A
```

U # >= 2, V #=< U-1. de '=' ici n'est pas comme en langage C !

```
U in 2..sup,     % U est un entier dans 2.. infini
V#=<U+ -1.       % et V =< U-1
```

B #\= 5.

```
B in inf..4\/6..sup.      % B à valeur dans -infini..4 U 6..infini
```

On reprend les exercices en séance. . . . /...

☞ A ce stade, à travers l'opérateur '=', il est important d'avoir compris l'unification et son fonctionnement qui est central à Prolog. De la compréhension de cette unification découle celle de "\=" vu plus haut et d'autres opérateurs utilisant des contraintes.
Le prédicat "dif/2" mérite également notre attention ! Il remplace avantageusement "\=".

☞ Voir section **??** en page **??** de ce document et le cours chapitre-1.

3 Travail 1 à compléter et à rendre (Famille)

Travail personnel à rendre (sous 15 jours) : Base de données familiales

Soient les informations suivantes (pas de majuscules) : fichier fourni

- Relation **epouse(Femme, Mari)**
  - sylvie est **l'épouse** de jean (ne pas utiliser les accents en Prolog).
  - marie est l'épouse de paul.
  - helene est l'épouse de jacques.
  - bernadette est l'épouse de jose et valentine celle de loic.

- Relation **enfant(Enfant, Père) et enfant(Enfant, Mère)**
  - jean est le **père** de jacques, pierre et marie.
  - pierre est le père eric et thomas.
  - jacques est le père vincent, loic et michèle.
  - jose est le père valentin et olivier,
  - marie est la **mère** de mark et de laure.
  - sylvie est la mère de sophie et d'alex.
  - thomas est le père de david et christine.
  - laure est la mère de Serge et Juliette.

Ci-dessous, on a l'arbre généalogique partiel de cette famille (rappelé plus loin en plus grand).

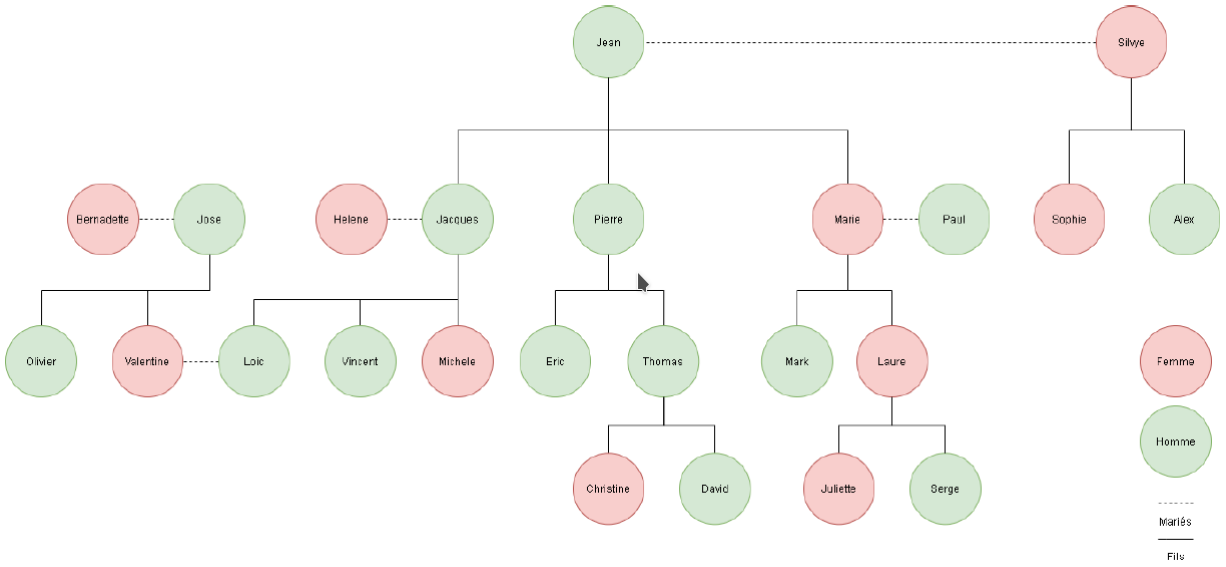


Figure 1: Arbre de la famille

Questions :

I- Écrire le prédicat **epoux(Mari, Femme)** que vous déduirez de la relation **epouse(Femme, Mari)**.

II- Sachant que pour un enfant E, un père P et une mère M,
E est enfant d'un père P si enfant(E,P) est donné
E est enfant d'une mère M si enfant(E,M) est donné

Compléter (à la main) la base de données et compléter le prédicat **enfant/2** (enfant à 2 paramètres) de sorte que :
E est enfant de M si M est épouse de P.
E est enfant de P si P est l'époux de M.

☞ Ne pas ajouter des doublons. N'ajouter enfant(X,Y) que s'il n'est pas déjà dans la base de données. Vous devriez ajouter :

alex et sophie pour jean
jacques, marie et pierre pour sylvie
enfant(laure et mark pour paul
loic, michele,helene et vincent pour helene
olivier valentine pour bernadette

III- A l'aide du graphe ci-dessus, ajouter le prédicat **homme/1** pour chaque individu (masculin de couleur verte dans le graphe) à cette base.

Vous devriez ajouter (les hommes) :

jean, jose, jacques, pierre, paul, alex, olivier, loic, vincent, eric, thomas, mark, david, serge

homme(jean). homme(paul). homme(pierre). homme(jacques). homme(mark). homme(thomas). homme(vincent). homme(eric). homme(loic). homme(david). homme(serge). homme(jose) homme(alex). homme(olivier).

IV- Ajouter la règle ('\+' veut dire not : la négation en Prolog)

femme(X) :- nonvar(X), \+ homme(X). % ce qui n'est pas 'homme' est 'femme'.

nonvar(X) vérifie que X a reçu une valeur avant que l'on vérifie s'il est un homme ou pas.
Faute de ce test, vous pourrez déduire que n'importe quel **Y** absent de notre Base de connaissances est une femme (puisqu'on ne peut pas prouver que **Y** est un homme!)
En logique, ce "phénomène" est appelé "negation as failure" :

V- Écrire les prédicats basiques:

Prédicat	à l'aide de(s) prédicat(s)
pere(Dad, Enf)	enfant/2 + homme/1 ou epouse/2
mere(Mam, Enf)	enfant/2 + femme/1 ou epoux/2
parent(Par, Enf)	si *Par* est le père ou la mère de Enf
fils(Enf, Par)	enfant/2 + homme/1
fille(Enf, Par)	enfant/2 + femme/1

VI- Quelles personnes satisfont la relation décrite ci-dessous pour 'loic' :

Qui est "le frère du fils du mari de la mère du père de la soeur du fils de la mère de **loic***" ?*

Et pour **alex** ?

N.B. : trouver un lien de parenté entre deux individus :
Sans autre traitement particulier, Prolog ne peut pas trouver qu'une description comme ci-dessus correspond à tel lien de parenté.
Plus simplement, si on sait que jean est un homme (on a le fait *homme(jean).*), on ne peut demandee *R(jean)* et obtenir *R=homme*.

Prolog n'accepte pas qu'un nom de relation soit une variable comme dans **Rel(alex, marie)** car on serait alors dans le cas d'une logique du 2e ordre (prédicats portant sur des prédicats où une relation peut être une variable). La logique du 1er ordre concerne des prédicats (ou relations) sur des individus (variables, constantes), pas sur les relations.

Par contre, on peut chercher une relation telle que l'équivalence ou l'implication.
Par exemple, pour l'équivalence, il est tout à fait possible de créer un programme qui ferait le travail en se basant par exemple sur la règle (naïve) que deux prédicats qui ont le même ensemble (exhaustif) de réponses sont EQUIVALENTS : chacun implique l'autre. De même pour l'implication.

VII- **Pour se détendre :**

Un jeune homme (pedro) se marie avec une dame nommée "brigitte". Ensemble, ils ont un enfant (un garçon nommé "yves").
Quelque temps après, le père de pedro ("yannick") rencontre "elisabeth" la fille de "brigitte". Ils se marient également et ont un fils "luc".
En admettant qu'un beau-fils est comme un fils et un beau père comme un père, pouvez-vous démonter que "pedro" est son propre grand-père !

• Voir aussi **Note 3** ci-dessous sur 'cut'.

Au besoin, vous pouvez ajouter d'autres faits (*épouse* / *enfant*) et règles à cette base en respectant l'énoncé.

4 Travail 2 à rendre

Cet exercice porte sur les termes Prolog (qui sont en fait des arbres).
L'enseignant fera un rappel sur les termes Prolog ainsi que sur les points suivants.

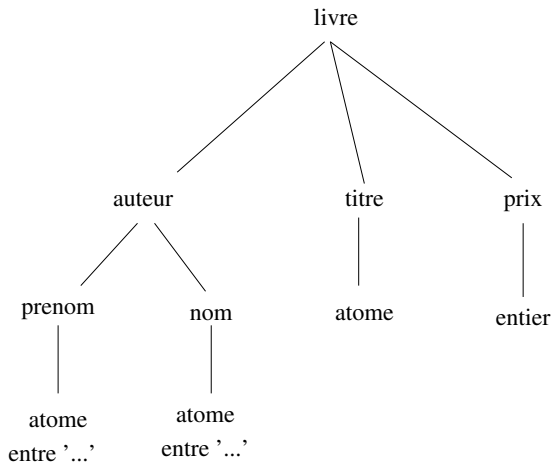
Points Prolog : Terme composé, Unification complète, Résolution, variable anonyme (pour les détails de l'unification, voir Note 6 des "Remarques et notes" ci-dessous).

Étant donné une base de faits comportant des termes composés, des questions exprimées en français sont traduites en Prolog.

N.B. : Prolog est un langage non typé : dire que le prix est un "entier" sert le concepteur de la base "livre"; Prolog ne sait tenir compte de ce typage (mais il nous est possible de tester dans notre programme si l'on a un entier à cet endroit).

☞ Des apostrophes autour d'un terme rend ce terme *atomique* (non décomposable comme dans le prénom composé *'jean pierre'*). Notons également que les guillemets sont utilisés pour et dans les chaînes de caractères (comme dans "Hello ici ECL").
Notez que la chaîne "l'ECL" s'écrit comme dans "Hello ici l'ECL"

Dans la BD. suivante, un livre est un terme composé (donc un arbre Prolog) dont la structure est donné par :



I- Soit la base de connaissances suivante :

% **guillemets** *à l'intérieur d'apostrophe (comme dans 'l"espion')*

livre(auteur(fredric, dard), titre('l"espion'), prix(45)).

% **apostrophes** *pour inclure des espaces dans un nom/ titre (comma dans 'les misérables')*

livre(auteur(victor, hugo), titre('les misérables'), prix(300)).

livre(auteur(eugene, delonay), titre('pendule de foucault'), prix(125)).

livre(auteur(victor, hugo), titre('fantasia chez les ploucs'), prix(200)).

livre(auteur(eugene, chang), titre('le silence'), prix(180)).

livre(auteur(umberto, ecco), titre('le nom de la rose'), prix(185)).

livre(auteur(umberto, ecco), titre('pendule de foucault'), prix(195)).

☞ Rappelez-vous : n'écrivez pas le fait :
livre(auteur(Victor,Hugo), ...

Car vous aurez placé DEUX VARIABLES (*Victor* et *Hugo*) dans ce fait. En d'autres termes, vous acceptez que n'importe qui ait écrit "les misérables" !

II- Formuler en Prolog les questions suivantes :

1. *Quel est le nom d'un auteur de livre dont le prénom est victor?* (voir Note ci-dessous)

   Réponse : *?- livre(auteur(victor, Nom),_,_). % Var. anonymes pour le titre et le prix*

   → *Nom = hugo*

   → *Nom = hugo*

   deux réponses car "victor hugo" apparaît deux fois dans la base.

2. *Existe-t-il un livre valant 40 Euros ?*
3. *Quel est le titre d'un livre de fredric dard ?*
4. *Quel est le prix du livre "le silence" ?*
5. *Quels sont les livres d'umberto ecco ?*
6. *Existe-t-il des livres du même titre écrits par des écrivains différents ?*

   ➞ N.B. : X \= Y : X différent de Y.
7. *Y a-t-il des livres dans la base avec un prix supérieur à 200 ?*
8. *Écrire une règle permettant de connaître les livres écrits par un écrivain donné E dont le prix est égal à P.*
9. *Quels sont les livres que l'on peut acheter avec 400 euros ?*

Note sur cut (!) :

Reprenons la dernière question : *Y a-t-il des livres dans la base ?*

?- livre(Livre, Titre, Prix).

→ *autant de réponses que de livre.*

Pour éviter les multiples réponses, on se sert de CUT (noté '!', voir le cours; l'enseignant vous donnera également des indications).

La requête suivante se contente de la première réponse :

?- livre(Livre, Titre, Prix), !.

→ *une seule réponse seulement.*

➞ Ce 'cut' signifie : dès que la question a une bonne réponse, ne plus chercher d'autres réponses.

On peut aussi poser la question par : *livre(_, _, _), !.*

5 Remarques et notes

☞ Sous SWI-Prolog, une requête portant sur un prédicat non défini prolog une erreur.
Par exemple : la question
che_pas_moi.
Donne une erreur en précisant que "che_pas_moi" n'est pas défini.

Si vous voulez obtenir un échec à la place d'une erreur, placez
:- dynamic che_pas_moi/0.
au début de votre fichier de code.

Une autre solution est de placer au début du fichier de votre programme :
:- set_prolog_flag(unknwon, fail).
Mais ceci est déconseillé par ses effets imprévisible sur le mécanisme interne de Prolog (dixit le manuel de référence de SWI-Prolog)

Note 1 : dans une question composée (conjonctive), les littéraux sont séparés par une virgule. Par exemple, une conjonction :

?- mortel (X) , eternel (X) . % existe-t-il une valeur 'c' pour X telle que mortel(c) ET eternel(c).

Il faut bien noter que les deux occurrences de la variable 'X' **représentent la même valeur**. Dès que la première X aura reçu une valeur, cette valeur sera propagée à la seconde X.

- Le "ou" (disjonction) s'exprime avec un point-virgule :
?- mortel (X) ; eternel (X) . % existe-t-il une valeur 'c' pour X telle que mortel(c) **OU** ...

Il faut noter que les deux occurrences de la variable 'X' dans cette disjonction **ne représentent pas la même valeur**. Autrement dit, cette requête donne les mêmes résultats que :

?- mortel (X) ; eternel (Z) . *OU*

% existe-t-il une valeur 'd' pour Z telle que mortel (d)

- Bien entendu, à l'intérieur d'une règle, les liens entre les variables de la partie gauche et celle de la partie droite imposent des contraintes éventuellement différentes (voir cours).

- Si la question disjonctive ne contient pas de variable, Prolog répondra par "**true**" au lieu de "**yes**".

Exemples :
- en posant la question

?-entree(salade) ; plat(poisson).
→ on obtient 'true' au lieu de 'yes' avec des possibilités habituelles de re-satisfaction (avec ';'). Pour simplifier, disons qu'il y a succès dans les 2 cas.

- en posant la question :

?-entree(salade) ; plat(poisson); chepas. *% "chepas" non défini*

→ on obtient les mêmes réponses en terminant sur un warning pour cause d'absence du prédicat 'chepas'.

- en posant la question :
?-entree(X) ; plat(poisson).
→ on obtient (comme d'habitude) des valeurs pour X avec 'yes/no' final.

Note 2 : l'utilisation de la négation notée **\+** :
Exemple : **Y-a-t'il un repas avec du 'poulet' en plat sans 'glace' au dessert ?**

Cette question peut-être formulée de différentes manières :
1- *?- repas(E, poulet, D), D \= glace. % D différent de 'glace'*

→ ....

2- *?- repas(E, poulet, D), D \== glace. % D différent de 'glace'*

→ .... idem

3- *?- repas(E, poulet, D), \+ (D = glace). % NON (D égal 'glace')*

→ .... idem

Mauvaise formulation de la question : si on peut prouver que *repas(_, poulet, glace)* **n'est pas** prouvable, on en déduira que le menu le propose pas. La formulation suivante :

4- *?- \+ repas(E, poulet, glace). % pas de repas avec poulet et glace ?*

→ no

échoue, c'est à dire que le menu le propose. Pour autant, cela ne veut pas dire qu'un repas avec 'poulet' mais sans 'glace' n'est pas possible (par exemple, repas(_, poulet, raisin) peut convenir).

Ce dernier cas présente donc une interprétation différente (erronée) de la question que l'on se pose (voir le cours sur la négation).

Note 3 : comment fait la négation (notée \+)
La négation d'un prédicat avec variables ne donne pas de valeur aux variables. Pourquoi ?

La raison est dans la définitions même de 'not' :

```
not(Predicat) :-
Predicat , ! , fail.    % Si 'Prédicat' est vrai, échouer !
not(Predicat).          % Sinon, réussir (sans chercher à prouver 'Prédicat').
```

Sachant que le "fail" "défait" les valeurs des variables, la première clause annule les instanciations des variables et la deuxième ne les affecte pas.

=> in fine, aucune variable éventuelle de 'Prédicat' ne reçoit de valeur.

Encore une remarque :

On peut mettre à profit ce fonctionnement pour tester le succès d'un prédicat sans affecter ses variables :

```
\+ \+ homme(X)
```

permet de savoir s'il y a au moins une clause 'homme(...)' définie (test de l'existence) sans pour autant donner une valeur à X.

Note 4

- Pour tester la différence de deux termes, on utilise les opérateurs **\=** et \==.

- L'opérateur =\= est utilisé pour la différence entre les valeurs de deux expressions arithmétiques (voir plus bas).

- Pour l'égalité des termes, on utilise = et **==**

- \+*(X=Y)* est équivalent à *X* \= *Y* (pour tester l'inégalité de X et de Y).

- Voir aussi le BE suivant.

Note 5 sur la logique du 2nd ordre :

Pas de question sous la forme *?- X(salade)* ni *?- A=entree, A(salade) .*

Mais *?- A = entree(salade), A.* est utilisable (l'appel de A est en fait *call(A)* et *A=entree(salade)*).

Voir aussi les manipulations des termes composés (arg, functor, ...).

Note 6 : rappel de l'unification

Unifier deux termes Prolog T1 et T2 (version simplifiée):
- Si T1 et T2 sont textuellement (littéralement) identiques Alors les 2 s'unifient

- Si T1 est une variable Alors associer T1 à T2 (T1 prend la valeur de T2)

- Si T2 est une variable Alors associer T2 à T1 (T2 prend la valeur de T1)

- Si T1 est de la forme f(A1,...An) et T2 : f(B1,...,Bn) (même symbole f et même n)

Alors unifier (récursivement) chacun des sous-termes Ai avec Bi en propageant les valeurs prises par les variables à chaque étape i aux restes des 2 termes.

L'unification produit un ensemble de couples .

Exemple :
Unifions*livre(X, titre('book'), prix(100))* avec

livre(auteur(Nom, jean), titre(Titre), prix(Pr))

=> On a : 'livre' = 'livre' et les deux termes ont le même nombre d'arguments (3)

=> Résultat : X = auteur(Nom,jean) et Titre = 'book' et Pr=100

Note 7 Quelques définitions utiles :

- Un fait ou une règle est appelé une **clause**. Une clause possède une partie gauche et une partie droite.

- Un **fait** est une clause (ou règle) sans partie droite (plus exactement, cette partie droite est 'true').

Ci-dessus, mortel(X) :- homme(X) est une clause (appelée aussi une **règle**).

Le fait homme(aristote) n'a pas de partie droite.

- Dans ces exemples, mortel(X) ou homme(aristote) sont appelés des littéraux. Un fait est donc réduit à un seul **littéral**. Un question (requête) est une règle sans partie gauche (plus exactement, sa partie gauche est 'false').

- Un paquet de clauses (paquet de règles et de faits) du même nom est appelé un **prédicat** (par exemple, le prédicat homme). Habituellement, on parle du prédicat **homme/1** pour désigner les clauses homme avec **un** paramètre. Ce qui permet aussi de désigner plus précisément le paquet de clauses qui nous intéresse.

- Il n'y a pas d'ordre entre les prédicats. Dans l'exemple ci-dessus, le prédicat mortel/1 peut être défini avant ou après homme/1. Par contre, l'ordre entre les clauses est important.

- Dans le littéral f(X, g(Y)), f et g sont appelés **symboles fonctionnels**. f est (mais pas g) également appelé **symbole prédicatif** si f est un prédicat .

- Un **programme** Prolog (logique) est constitué d'un paquet de prédicats. Un programme représente un ensemble de théories et les questions posées seront des théorèmes à démontrer.

On ne décrit pas de fonctions (au sens général) en Prolog mais des prédicats. Eviter de penser qu'en Prolog, on peut écrire une fonction qui renvoie par exemple un réel ! Un prédicat Prolog renvoie TOUJOURS true ou false, rien d'autre.

De même, Prolog n'évalue des termes qu'à la demande explicite. En Prolog, 1+2 ne vaut pas 3 tant qu'on n'a pas demandé son évaluation (par exemple, par is).

Note 8 : Réfutation et Résolution

- Pour démontrer un théorème (une requête Prolog) **R** étant donné un programme **P**, on veut savoir si **P - R**.

Autrement dit, on veut savoir si **P /\R - true**.

Pour ce faire, Prolog démontre que **P /\˜R - false ,** c'est à dire que la réfutation de R dans P mène à une contradiction.

NB : l'opérateur **-** est une déduction formelle (théorique). Son équivalent calculatoire en Prolog est **=** .

Prolog utilise modus tolens (A -> B /\˜B - ˜A) ainsi que la réfutation (si **P /\˜R - false** alors **P - R**.

En particulier, si A=true, on aura, (B /\˜B - false).

6 Installation Prolog

☞ Privilégiez la 1e manière : il y a un éditeur intégré qui vous facilitera la vie !

6.1 La 1e manière : SWI-Prolog

Certains d'entre vous (en particulier les possesseurs de MacOs dernière version) ont déjà procédé à cette installation.
☞ Lisez quand-même. Il ya un élément sur un éditeur intéressant.

1. Télécharger swi-prolog : https://www.swi-prolog.org/download/stable selon votre plateforme.

   Selon votre plateforme, procédez à l'installation :

   ◦ **Windows** : exécuter swipl-xxx-64.exe. Cela installera Prolog comme une application windows quelconque.

   ◦ **Mac** : ouvrir swiplxxx.dmg et procédez à l'installation dans le dossier Applications

   ◦ **Linux** : sudo apt install swi-prolog

2. Lancez l'exécutable *swi prolog* (selon les plateformes) :

   ◦ Windows et Mac : lancez l'exécutale **swipl** (vous savez lancer une application installée sur votre plateforme !)

   ◦ Linux : dans un efenetre "terminal" (console), lancez l'exécutale **swipl**

3. Taper la requête **set_prolog_flag(editor, pce_emacs).**

   ➛ Nous aurons besoin une seule fois d'exécuter cette requête pour dire notre choix de l'éditeur.

   ➛ Lors des prochaines utilisations, Prolog s'en souviendra.

4. Plus de documentation et aide sur https://www.swi-prolog.org/pldoc/man?section=pceemacs.

5. D'une manière générale, vous pouvez sélectionner l'éditeur que vous voulez (à la place de *pce_emacs*).

   ☞ Par exemple, sous Windows, vous pouvez également demander **set_prolog_flag(editor, wordpad).**. Voir également plus bas si votre éditeur n'est ni *pce_emacs* ni *wordpad*.

6. Quelque soit votre plateforme, sous Prolog, vous disposez des commandes 'cd', 'pwd', ...

   Par exemple, pour savoir dan srépertoire vous-êtes, taper :

   ```
   pwd.        % le répertoire courant
   ```

   Pour vous placer dans votre répertoire usuel (répertoire de votre compte) taper :

   ```
   cd('~/')        % vous placera dans votre répertoire principal
   ```

7. Sous Prolog, taper la requête suivante pour éditer / créer le fichier 'prog1.pl'.

   edit(file('prog1.pl')). % ➛ true : la réponse de swi

   Cette requête lance l'éditeur. Il s'agit d'un éditeur élémentaire mais utile.

   ☞ Cette requête crée le fichier 'prog1.pl'. Si 'prog1.pl' existe déjà, taper simplement **"edit('prog1.pl')."**. Néanmoins, *edit(file('prog1.pl')).* fonctionnera toujours !

8. Sur MacOs, si l'exécution de cette dernière commande échoue, c'est parce que vous n'avez pas déjà installé tous les modules nécessaires sur votre machine.

   Dans ce cas, faites comme à la première séance : utilisez un éditeur quelconque hors Prolog puis consultez votre programme. Par exemple, utilisez l'éditeur 'textmate' (version gratuite existe); éditez 'prog1.pl'; sauvegardez et sous prolog, chargez le programme par `['prog1.pl']`.

9. De même, si vous avez sélectionné un autre éditeur que *pce_emacs* ou *wordpad* (p. ex, si votre éditeur est 'vi' sous Mac ou Linux), après l'édition et la sauvegarde de votre fichier 'prog1.pl', une fois revenu sous Prolog, vous serez peut-être obligé de consulter votre programme par la requête `['prog1.pl']` pour le charger explicitement.

10. Sous l'éditer, taper les lignes suivantes (pour tester) :

    mortel(X) :- homme(X).

    homme(socrates).

    homme(aristote).

    homme(man).

11. Sauvegardez le fichier (Menu File / Save)

12. Dans le menu de l'éditeur, sélectionner "compile buffer" (ou "build").

    ➛ Cela charge votre programme sous Prolog.

    ➛ Si vous avez des erreurs, elles sont signalées; corrigez !

13. Maintenant, taper la requête

    `mortel(X).`

    ➛ Pour avoir les autres réponses, appuyez sur ";".

    ➛ Un retour-chariot (touche "entrée") arrête les réponses.

14. Tester le mode trace en tapant "trace." puis retaper la requête `mortel(X).` pour avoir une trace.

15. Terminer la trace avec "notrace".

16. Pour quitter Prolog, on tapera la commande "**halt.**" .

Deux remarques (utiles pour plus tard; voir avec l'enseignant) :
1) Pour avoir accès aux contraintes du domaine fini, il faut importer la librairie concernée. On exécutera la requête

```
use_module(library(clpfd)).
```

D'une manière plus générale, insérer au début de vos fichiers Prolog la requête

```
:- use_module(library(clpfd)).
```

2) Par défaut, si vous essayez d'exécuter une requête qui appelle un prédicat inexistant, vous aurez une erreur. Par exemple, toujours sous Prolog, taper la requête

```
femme(X).        % Vous aurez un message d'erreur "prédicat inexistant"
```

Si vous voulez obtenir un échec (*fail*) à la place de l'erreur, il faudra insérer au début de votre fichier de programme :

```
:- current_prolog_flag(unknown, fail).
```

➛ On ne peut exécuter la même requête sous Prolog (on aura un refus). Par contre, vous pouvez réaliser le même effet en utilisant la requête prédéfini `dynamic` .

3) La négation du prédicat `p` se dit `not(p)` ou $\backslash + p$

6.2 Editer un programme

Le code source du premier programme est donné ci-dessous.
Comme pour tout autre langage, on doit créer un programme a l'aide d'un éditeur puis charger ce programme sous Prolog. Ici, pour apprendre, on utilise Prolog comme un **interpréteur** (même s'il peut compiler des programmes, on verra cet aspect plus loin).

Si vous travaillez sous Linux :

- On peut utiliser différents éditeurs tels que " kate ", " kwrite ", "vi", "emacs", "gedit", "kedit", ...
- Choisissez un éditeur, puis lancez-le dans une fenêtre LINUX.

Par exemple, par la commande Unix :

kate menu.pl & *(kate est moins encombrant et gère mieux les fenêtres)*

Rappel : le '&' placé à la fin de la commande Unix "kate ..." permet de récupérer le prompt d'Unix pour exécuter d'autres commandes.

☞ Pour les Geeks ! : Si vous avez oublié le '&', taper dans la même fenêtre de lancement "contrôle-Z" puis "bg %1".

Si vous travaillez sous Windows :

- Vous pouvez utiliser votre éditeur habituel (*wordpad*, ... mais éviter *Word* ou *notepad*)

Si vous travaillez sous MacOs :

- *Textedit* est l'outil d'édition de base. Attention à ce que le fichier ne soit pas sauvegardé au format ".rtf". Pour cela, aller dans les préférences et demandez à enregistrer les fichiers en ".txt" puis redémarrez *textedit* .
Vous pouvez également travailler avec les éditeurs "vi", emacs", ... ou même celui de CodeBlocks !

Créons notre programme :
sous l'éditeur de votre choix, créer votre programme Prolog en copiant les prédicats donnés ci-dessous; **sauvegarder** sous le nom *menu.pl* (ne pas oublier '.pl').

➙ Ne pas quitter l'éditeur pour pouvoir ensuite y faire des modifications.

7 Annexes

7.1 Unification

Unifier deux termes t1 et t2 : trouver des valeurs pour les variables (de ces termes) qui rendent t1 et t2 **identiques**.

Principe de l'unification

Pour unifier deux termes t1 et t2 en Prolog, *on écrit* ***t1 = t2.***

1- Sans les termes composés

- Deux termes identiques s'unifient ;
- Une variable s'unifie avec n'importe quel autre terme ;
- Deux constantes différentes ne s'unifient pas.

Exemples :

12=15 → échec
X=eleve → succès, Θ={X/eleve}
X=Y → succès, Θ={X/Y}
X=X → succès, Θ={}
Ecole="ecole" → succès, Θ={Ecole/"ecole"}

2- Cas des termes composés termes composés

Soit deux termes T1 et T2 à unifier (noté **T1 = T2**).

➛ Si T1 et T2 s'unifient, cet algorithme produit une substitution Θ telle que $apply(\Theta, T1) = apply(\Theta, T2)$. On note cela par $\Theta(T1) = \Theta(T2)$ ou encore $T1_\Theta = T2_\Theta$.

- Θ initial ={}.
- L'unification des termes "simples" vue plus haut produira son Θ
- L'unification de deux arbres (termes composés) peut réussir si
  - les racines des 2 arbres sont identiques;
  - pour un niveau donnée, il y a autant de fils dans chaque arbre;
  - les fils respectifs s'unifient (par ce même algorithme)

Exemple : **personne(jean, Age) = personne(Y, 25)** produira Θ= {Age/25, Y/jean}
et Θ appliqué aux deux termes les rend identiques.

3-Cas général : unification de deux termes quelconques

On obtient donc l'algorithme général. Dans ce cas, l'unification de deux termes t1 et t2 consiste à trouver un unificateur le plus général Θ tel que $t_1\Theta = t_2\Theta$.
L'algorithme suivant (algorithme de Robinson) produit une substitution

La fonction ci-dessus renvoie un (ensemble de substitution) Θ. Si la valeur NULL est renvoyé (ce qui est différent de $\Theta = \{\}$), alors l'unification a échouée. Sinon, c'est un succès et Θ donne l'état des variables.

Fonction unifier(T1,T2 : termes; Θ : substitution) renvoie substitution

Cas

- *T1 est identique à T2 : retourne(Θ)*
- *T1 est une variable : retourne ($\Theta \cup$ {T1/T2})*
- *T2 est une variable : retourne ($\Theta \cup$ {T2/T1})*
- *T1 ou T2 est une constante : retourne(NULL);* *% cas d'échec*
- *T1 et T2 sont des termes composés : oit :* $T1 = f(t1, t2, ..., tn)$ *et* $T2 = f(t'1, t'2, ..., t'n)$.

  $i \leftarrow 0$

  Répéter

  $i \leftarrow i + 1$;

  appliquer Θ *à* t_i *et à* t'_i

  $\Theta \leftarrow unifier(t_i, t'_i, \Theta)$

  Jusqu'à ($i > n$) ou (Θ == NULL)
- *Autre :* $\Theta = NULL$

Fin cas

retourne(Θ)

Fin unifier

Un exemple : *on veut unifier T1 et T2*

T1 = f(a, g(X, b)) et **T2** = f(Y, g(b, X))

- étape 1- le symbole à la racine des 2 termes est identique (f) et les 2 termes ont 2 fils chacun.
- étape 2- **Y = a** (Y est-il unifiable avec a ?) ➙ C'est un cas simple et le résultat est Θ={Y/a}.

  On applique Θ à g(X, b) et à g(b, X) :

  - Θ(g(X, b)) = g(X, b) car X n'est pas lié dans Θ
  - Θ(g(b, X)) = g(b, X) idem

  A ce stade, avec $\Theta = \{Y/a\}$ et on a (l'égalité des vertes) f(a, g(X, b)) = f(a, g(b, X))

  On continue sur les "restes" des deux termes
- étape 3- **g(X, b) = g(b, X)** c'est à dire :

  3.1- **X = b** ➙ succès, modifie Θ qui devient Θ={Y/a, X/b}

  On applique Θ aux restes et on aura : Θ(X) = b, Θ(b) = b.

  A ce stade, avec Θ={Y/a, X/b}, on a (légalié des vertes)f(a, g(X, b)) $=_{3.1}$ f(Y, g(b, X))

  On continue sur les restes

  3.2- **b=b** ➙ succès et cette étape n'ajoute rien de plus à Θ;

  Le résultat de l'unification est substitution (non NULL) : Θ=**{Y/a, X/b}**

➛ L'unification a réussit et a produit Θ

7.2 Trace d'un programme

Pour activer le mode trace.

?- trace.

Le mode trace dévermine les prédicats d'un programme en surveillant 4 états (appelé trace box) d'un prédicat :

Call : appel (lancement) de la démonstration du prédicat
Fail : échec (après d'éventuels tentatives 'redo')
Exit : sortie avec succès
Redo : on ré-essaie la clause suivante dans le paquet de clauses du prédicat.

Pour désactiver le mode trace :

?-notrace.

Pour placer des mouchard sur un prédicat :

spy(nom-prédicat, arité) : pour placer un mouchard sur un prédicat
Exemple : ?- spy(dessert/1). Moucharder dessert à un argument.

Pour supprimer un mouchard :

nospy(nom-prédicat, arité) : enlever le mouchard sur 'nom-prédicat'.
Exemple : ?- nospy(dessert/1).

Ou

?-nospy.

Pour enlever tous les mouchards.

Les commandes de déverminage disponibles en mode trace :
Retour-Chariot : affichage étape par étape.
c : (creep) déverminage pas à pas (= Retour-Chariot)
l : (leap) continuer sans déverminage jusqu'au prochain point mouchard (placé avec spy).
s : (skip) ne plus déverminer jusque la fin (succès ou échec) du prédicat.
r : (repeat creep) le debug continue sans interactivité pas à pas.
h ou ? : (help) affiche les commandes disponibles.
t : (backtrace) afficher en backtrace en aboutissant à l'appel actuel (que c'est-il passé avant ?)
t i : (backtrace)afficher en backtrace depuis l'appel numéro i à l'appel actuel
u : (undo) défaire ce qui a été fait à l'appel actuel et le refaire.
u i : (undo) défaire ce qui a été fait à l'appel numéro i et le refaire.
< : remettre le niveau de trace à 10.
< d : remettre le niveau de trace à la valeur d.
a : (abort) avorter et arrêter l'exécution (de la requête).
n : (nodebug) arrêter le mode debug pour cette exécution (le mode trace reste actif).
+ : ajoute à l'appel courant un mouchard, en rapport avec la commande leap (" l ").
- : enlève le mouchard de l"appel courant, en rapport avec la commande leap (" l ").

Mise en œuvre de la trace :

il y a plusieurs manières de déverminer.

1- On place des point 'spy' sur le prédicat qui nous intéresse :

?- spy(dessert/1).

Puis on soumet une requête.

?- repas(A,B,C).

→ seul le prédicat principal 'repas' et le prédicat 'dessert/1' seront déverminés.

2- Si on n'a pas d'idée sur le ou les prédicats à déverminer, on active le mode trace puis arrivé sur le prédicat à surveiller, on place un mouchard à l'aide de '+'. On utilisera ensuite la commande leap (" l ") pour " sauter " la trace des autres prédicats mais s'arrêter là où on a placé un mouchard.
On peut procéder à l'inverse et supprimer (par '-') la trace sur les prédicats qui ne nous intéressent pas laissant le mode trace sur les autres.

3- On peut à tout moment utiliser 's' (skip) sur call/redo si on ne veut pas les détails d'appel sur un prédicat.

4- Si on désire seulement constater le niveau de descente (pour comparer deux prédicats similaires), on active le mode trace mais on utilise la touche " r " (repeat crep). Ainsi, on voit les trace mais sans interactivité.

7.3 Utilisation de Gprolog

☞ Si vous souhaitez utiliser cette version-là !

Procédez de la manière suivante (selon votre plateforme) :

Linux : Ouvrir une fenêtre terminale (sur le serveur CC06 de l'ECL) et devant le prompt de Linux (>, $, ...), taper

gprolog

Si vous utiliser Linux sur votre portable personnel, procéder comme pour MacOS (ci-dessous).

MacOS : Sous Mac OS, récupérer Gnu Prolog (sur le site d'INRIA en passant par Google); installer puis lancer l'application gprolog dans une fenêtre terminale en tapant :
textbf*gprolog*

Il existe une interface graphique (GUI) pour Gprolog appelée XGP que vous pouvez télécharger et installer en plus.
Dans ce cas, vous lancerez l'application XGP (dans votre dossier *Applications*).

Windows : Lancer l'exécutable **gprolog.exe** dans le répertoire L:\GNU-Prolog\bin.

Cette commande effectue le lancement de *GNU Prolog*.

Après lancement de gprolog selon votre plate-forme, un message de bienvenu suivi du prompt de Prolog s'affichent :

?-

La suite est la même que pour Prolog.

7.4 Quelques différences entre Prolog et Gprolog

- Différence du domaine des contraintes : Prolog utilise les entiers négatifs et positifs alorsque Gprolog exclue les entiers négatifs.
- On peut utiliser des contraintes sur les réels en Prolog (via lp_solve), pas sous Gprolog.

- L'éditeur en ligne de GProlog possède un mécanisme de complétion des identificateurs.
Par Exemple : si vous avez chargé un programme avec un prédicat tel que " mortel " (comme dans l'exemple de l'introduction), taper "*mort*" puis tabulation pour avoir *"mortel"* (voir aussi le manuel de Gprolog).

- Sous Gprolog, suite à une réponse, on peut demander toutes les solutions avec " a ". Prolog ignore cet aspect.
-Gnu Prolog sous *windows* dispose d'un petit menu.

- Avec GProlog, sous Windows, vous pouvez charger vos programmes par le menu.
- Sous Gprolog, on a le drapeau *set_prolog_flag(unknown, warning)*. Où l'absence d'un prédicat provoquera un avertissement + échec.

-Prolog utilise *dif(T1,T2)* sur deux arbres, pas Gprolog (dommage).

Contents