Université d'Aix-Marseille - L3 info - Logique : TP 1

1 Un exemple introductif

1.1 Base de faits

Une base de faits est un ensemble de faits que l'on déclare être vrais. Ces faits peuvent être stockés dans un fichier muni d'une extension `.pl`. Par exemple, le fichier `genealogie.pl` rappelé ci-dessous déclare des liens de parenté :

```
/*--------------------------------- fils(Pere, Mere, Fils) -*/
fils(jean, sophie, pierre).
fils(jean, sophie, jacques).
fils(jacques, sylvie, michel).
fils(michel, laurence, thomas).
fils(thomas, stephanie, alexandre).

/*------------------------------- fille(Pere, Mere, Fille) -*/
fille(jacques, sylvie, nathalie).
fille(francois, sylvie, mathilde).
fille(paul, virginie, stephanie).
fille(pierre, marie, virginie).
```

Vous remarquerez que nous n'utilisons aucune majuscule. La raison est qu'elles sont réservées aux variables : attention aux erreurs !

Pour charger cette base de faits, afin de pouvoir l'interroger, il suffit de taper dans un terminal : `$ swipl -s genealogie.pl`

1.2 Requêtes

Maintenant la base chargée, il est possible de poser des questions à Prolog sous forme de requêtes :
Testez les requêtes

```
?- fils(jean,sophie,pierre).
```

et

```
?- fils(jean,sophie,sophie).
```

Cherchons maintenant à savoir qui sont les fils de Jean et Sophie. Nous avons ici besoin d'une variable : elles sont toujours représentées par un identifiant débutant par une majuscule ou bien par le symbole _.

```
?- fils(jean,sophie, X).
```

Prolog répond Pierre, mais ne revient pas à l'invite car il a d'autres réponses à fournir : si vous voulez voir la prochaine réponse tapez " ;" ou "espace" (prolog vous répond Jacques), sinon tapez entrée.

Si on souhaite savoir s'il existe une personne satisfaisant plusieurs propriétés (une conjonction donc), il suffira de séparer ces propriétés d'une virgule. Par exemple, pour savoir si un fils de Jean et Sophie a eu une fille, il suffit de taper :

```
?- fils(jean,sophie, X), fille(X,Y,Z).
```

La réponse de Prolog est

```
X = pierre,
Y = marie,
Z = virginie
```

Si on ne s'intéresse qu'à la valeur de X, on posera plutôt la question suivante :

```
?- fils(jean,sophie, X), fille(X,_,_).
```

La réponse de Prolog est `X = pierre`. Vous pouvez quitter Prolog en tapant

```
?- halt.
```

Si vous souhaitez recharger une nouvelle base de faits contenue dans un fichier `base.pl`, sans avoir à quitter et relancer Prolog, il suffit de taper

```
?- consult('chemin/base.pl').
```

ou `?- consult(base).` si le fichier est dans le répertoire courant.

1.3 Ajout de règles

Nous allons maintenant enrichir notre base de faits par des règles (données sous forme de clauses de Horn), permettant de définir des relations (ou prédicats).

Par exemple, pour définir la relation `enfant(Pere, Mere, Enfant)` (notez que ici `Enfant`, `Pere` et `Mere` sont des variables) il nous suffit d'ajouter à la base de faits les règles suivantes :

```
enfant(X, Y, Z) :- fils(X, Y, Z).
enfant(X, Y, Z) :- fille(X, Y, Z).
```

En terme de logique du premier ordre, cela correspond à la formule $\forall X, Y, Z((fils(X, Y, Z) \Rightarrow enfant(X, Y, Z)) \wedge (fille(X, Y, Z) \Rightarrow enfant(X, Y, Z)))$

Voici quelques exemples assez explicites pour ne pas nécessiter d'explication :

```
/*------------------------------------------- homme(X) -*/
homme(X) :- fils(_,_,X).
homme(X) :- fils(X,_,_).
homme(X) :- fille(X,_,_).

/*------------------------------------------- femme(X) -*/
femme(X) :- fille(_,_,X).
femme(X) :- fils(_,X,_).
femme(X) :- fille(_,X,_).

/*------------------------------ parent(Enfant, Parent) -*/
parent(Enfant, Pere) :- enfant(Pere, _, Enfant).
parent(Enfant, Mere) :- enfant(_, Mere, Enfant).
```

Ajoutez ces règles à votre base de faits, testez quelques requêtes sur cette nouvelle base. Notez l'utilisation des `_` : il s'agit de variables anonymes qui remplacent les variables qui n'ont qu'une occurence dans une règle et dont on ne souhaite pas connaàtre la valeur d'instanciation. Prolog met un warning quand une variable n'a qu'une occurence dans une règle car elle pourrait être anonymisée, donc il soupçonne qu'il s'agit d'une variable qui a plusieurs occurences et qu'on a ponctuellement mal orthographié (essayez par exemple de remplacer `homme(X) :- fils(_,_,X).` par `homme(X) :- fils(Y,Z,X).`).

Ajoutez maintenant les règles décrivant les prédicats suivants :

`grand_parent(PetitEnfant, GrandParent)` et
`grand_pere(PetitEnfant, GrandPere)`.

Ecrivez maintenant le prédicat `frere(X, Frere)` *définissant que X et Frere ont le même père. Lancez la requête* `?- frere(pierre,X)`.

Vous remarquez que Prolog répond que Pierre est un frère de Pierre. Corrigez votre définition du prédicat en ajoutant l'assertion précisant que X et Frere doivent être différents : `X \= Frere`. Notez qu'en prolog `t1=t2` correspond à l'unification de `t1` et `t2`. Donc `X \= Frere` signifie que `X` et `Frere` ne sont pas unifiables.

Remarquez la définition suivante du prédicat `ancetre` qu'on pourrait qualifier de *récursive*. Lisez bien l'introduction à Prolog qui vous a été fournie pour éviter de créer des requêtes trop lentes ou qui bouclent indéfiniment.

```
/*------------------------ ancetre(Descendant, Ancetre) -*/
ancetre(Descendant, Ancetre) :- parent(Descendant, Ancetre).
ancetre(Descendant, Ancetre) :- parent(Descendant, Y), ancetre(Y, Ancetre).
```

Ces règles correspondent à la formule suivante :

$$\forall D, A, Y((parent(D, A) \Rightarrow ancetre(D, A)) \wedge ((parent(D, Y) \wedge ancetre(Y, A)) \Rightarrow ancetre(D, A)))$$

Lancez une requête pour déterminer les ancêtres d'Alexandre.

2 Exercice : Cursus post-bac

On veut modéliser les cursus post-bac, en indiquant si le passage d'une année à l'autre est de droit ou par sélection. Pour ceci, on définit les prédicats

- `droit(A1, A2)`, qui est vrai quand on passe de droit de l'année A1 à l'année A2 (quand on a obtenu son année A1),
- `selection(A1, A2)`, qui est vrai si le passage de l'année A1 à l'année A2 se fait par sélection,
- `bacplus(A, N)`, qui est vrai si l'année A correspond à un niveau bac+N, et
- `diplome(A, D)`, qui est vrai si à l'issue de l'année A on obtient le diplôme intutilé D.

L'ensemble de ces prédicats est défini dans la base de faits `cursus.pl`. Notez que les entiers sont représentés par des termes définis par induction : 0 est un entier et si le terme X est un entier alors le terme s(X), son successeur, est un entier (ce sont les entiers dits de Peano).

1. *Écrire le prédicat* `temps_minimum_bac_diplome(D, T)`*, qui est vrai si le diplôme D s'acquiert au minimum en T ans après qu'on ait passé le bac.*
2. *Écrire le prédicat* `passage(A1, A2)` *qui est vrai s'il est possible de passer directement de l'année A1 à l'année A2 (de droit ou par sélection).*
3. *Écrire le prédicat* `parcours(Debut, Fin)`*, qui est vrai s'il existe une faàon d'aller de l'année Debut à l'année Fin en passant par une suite d'années.*
4. *Écrire le prédicat* `parcours_selectif(Debut, Fin)`*, qui est vrai s'il existe un parcours de Debut à Fin tel qu'au moins un passage d'une année à une autre se fait par sélection.*
5. *Écrire le prédicat* `temps_parcours(Debut, Fin, T)`*, qui est vrai si on peut aller de l'année Debut à l'année Fin en T ans.*

3 Opérations sur les entiers de Peano

On considère les entiers de Peano représentés inductivement par des termes : 0 est un entier et si le terme X est un entier alors le terme s(X), son successeur, est un entier :

$$\frac{}{0} \qquad \frac{N}{s(N)}$$

1. *Écrire le prédicat* `entier(X)` *qui est vrai si* `X` *est un entier. Lancez une requête pour obtenir quelques entiers.*
2. *Écrire les prédicats* `pair(X)` *et* `impair(X)` *qui sont vrais respectivement si* `X` *est pair, si* `X` *est impair.* On utilisera le fait que si `X` est pair alors `s(s(X))` aussi. *Lancez les requêtes pour obtenir quelques entiers pairs ou impairs.*
3. *Écrire les prédicats* `inf(X, Y)` *et* `infeq(X, Y)` *qui sont vrais respectivement si* `X` *est strictement inférieur à* `Y`*, si* `X` *est inférieur ou égal à* `Y`*.* On utilisera le fait que si `X` est inférieur à `Y` alors `X` est inférieur à `s(Y)`. *Lancez les requêtes pour obtenir les entiers strictement inférieurs à* `s(s(s(0)))`*.*
4. *Écrire les prédicats* `add(X, Y, Somme)`*,* `sub(X, Y, Difference)`*,* `mul(X, Y, Produit)` *et* `div(X, Y, Quotient)` *qui permettent les opérations d'addition, de soustraction, de produit et de quotient entre* `X` *et* `Y`*.* On pourra utiliser le fait que `X+(Y+1) = (X+Y)+1`, que si `X + Y = Z` alors `Z - X = Y`, que `X*(Y+1) = X*Y + X`.

   Vos clauses doivent au moins permettre d'obtenir le bon résultat de l'opération si `X` et `Y` sont instanciés. *Lancez diverses requêtes pour tester le comportement de vos prédicats en fonction des arguments instanciés et non instanciés. Vérifiez que certaines requêtes posent problème.* Ex : il est tout à fait possible (et acceptable) que `?- mul(X, Y, s(s(s(s(0))))).` ne permette pas d'obtenir les 4 réponses qu'on pourrait attendre.

   Vous pourrez regarder comment la résolution est effectuée en lançant le mode trace : `?-trace.` que vous pourrez ensuite quitter par la commande `?-notrace.`
5. *Écrire les prédicats* `puissance(X, Exposant, Resultat)` *et* `fibonacci(N, Resultat)` *qui permettent les calculs de* `X` *à la puissance* `Exposant` *et du* `N`*-ième nombre de Fibonacci.*

4 Les listes

Une des structures de données fondamentale de Prolog est la liste définie par l'induction suivante :

$$\frac{}{[]} \qquad \frac{T}{[H|T]}\ \texttt{H quelconque}$$

Le symbole `H` désigne la tête (head) de la liste, donc son premier élément, le symbole `T` désigne la queue (tail) de la liste, donc la liste qui suit la tête. Notez que `H` peut être de n'importe quel type (nous verrons quelques types prédéfinis plus tard). Une liste pourra aussi être donnée par énumération de ses éléments : `[abc, 1, f(x), Y]`. Référez vous au poly pour plus de détails.

1. *Écrire le prédicat* `appartient(E, L)` *qui est vrai si* `E` *appartient à la liste* `L`*.*
2. *Écrire le prédicat* `longueur(L, N)` *qui est vrai si* `N` *est la longueur de la liste* `L`*.* L'entier `N` est un entier de Peano : `s(s(...0...))`.
3. *Écrire le prédicat* `concat(L1, L2, L)` *qui est vrai si la liste* `L` *est la concaténation des listes* `L1` *et* `L2`*.*
4. *Écrire le prédicat* `supprime(E, L, L1)` *qui est vrai si la liste* `L1` *est la liste* `L` *privée d'une occurence de l'élément* `E`*.*
5. *Écrire le prédicat* `inverse(L, L1)` *qui est vrai si la liste* `L1` *contient les mêmes éléments que* `L` *mais dans l'ordre inverse.*
6. *Écrire le prédicat* `tri_naif(L, Lt)` *qui est vrai si* `Lt` *contient les entiers (de Peano) de* `L` *dans l'ordre croissant. La définition de ce prédicat utilisera deux autres prédicats que vous écrirez aussi* :
   — le prédicat est `triee(L)` qui est vrai si `L` est triée dans l'ordre croissant.
   — le prédicat `permutation(L, Lp)` qui est vrai si `Lp` est une permutation de `L`.