# Des exercices de Prolog
# à destination d'étudiants de licence informatique en troisième année

Jean Lieber, Dmitry Sokolov, Université de Lorraine, `Prénom.Nom@loria.fr`

Dernière version : 31 août 2020
Page du cours : `http://homepages.loria.fr/JLieber/cours/logiqueL3/`

**Remarques sur ce document**

— Ce document rassemble des exercices simples de Prolog qui n'utilisent qu'une petite partie de ce langage de programmation.

— Ce document contient probablement des erreurs (d'orthographe, de grammaire ou d'autres natures). Merci de le signaler à ses auteurs.

## Informations préliminaires

SWI-Prolog est installé sur les machines, voici un simple exemple d'utilisation :

1. Écrivez un programme Prolog sous la forme d'un fichier texte avec une extension .pl. Par exemple, le fichier `parent.pl` avec le contenu suivant : `parent(john, ann).` .
2. Ouvrez un terminal (Ctrl+Alt+T) et naviguez jusqu'au répertoire où vous avez enregistré votre programme.
3. Ouvrez SWI-Prolog en executant `swipl parent.pl`
4. Pour poser des requettes au programme chargé, tapez les buts et regardez le résultat, par exemple, si on tapait `parent(X,ann).` , l'inteprète nous donnerait `X = john.` .
5. Pour quitter SWI-Prolog, tapez `halt.` .

Il existe également un interprète en ligne : `https://swish.swi-prolog.org/` (qui fonctionne quand il veut bien).

## Exercices

**Exercice 1** La dernière question du dernier exercice de l'examen de décembre 2018 était la suivante :
(Début)
Soit $B = \{r, f_1, f_2\}$ la base de Horn avec $r = p(x,y) \wedge q(y,z) \Rightarrow q(x,z)$, $f_1 = p(a,b)$ et $f_2 = q(b,c)$.
Montrez par chaînage arrière que $B \models q(a,c)$.
Vous représenterez votre preuve par un arbre et-ou.
Montrez par chaînage arrière que $B \models_{\text{HMC}} \neg q(c,a)$, où $\models_{\text{HMC}}$ est la relation d'inférence sous hypothèse du monde clos.
(Fin)
Utilisez l'interprète Prolog pour déterminer que $B \models q(a,c)$ et $B \models_{\text{HMC}} \neg q(c,a)$.

---

**Exercice 2** Créez un fichier `famille.pl` et éditez-le en répondant aux questions suivantes :
**Q1.** Choisissez quelques personnes de votre famille (réelle ou imaginaire) et codez en Prolog des faits sous la forme de lien de parentés à l'aide des prédicats binaires `mere` et `pere`, par exemple

```
mere(marie, irene).
pere(pierre, irene).
```

**Q2.** Reprenez les règles du cours pour les prédicats `parent`, `grand_pere`, `grand_mere` et `ascendant` (voir section 3.7.5) et ajoutez-les au programme (on mettra les règles avant les faits).
**Q3.** Utilisez l'interprète Prolog pour répondre à différentes questions concernant votre famille.
**Q4.** Enrichissez la base de règles pour les relations `oncle_tante`, `cousin_e` et interrogez l'interprète (`oncle_tante(a, b)` indique que `a` est un oncle ou une tante de `b`; `cousin_e(a, b)` indique que `a` est une cousine ou un cousin de `b`).
**Remarque :** Pour cette dernière question, vous pourrez avoir besoin du connecteur `not` qui est proche du connecteur $\neg$ en logique du premier ordre [1].

---

**Exercice 3**
**Q1.** On considère les affirmations suivantes (dues à Lewis Carroll, dans *La logique symbolique*) :

A1 Tous les canards vivant dans ce village qui sont marqués d'un B appartiennent à Mrs. Bond.

A2 Les canards vivant dans ce village ne portent pas de col en dentelle, à moins qu'ils n'appartiennent à Mrs. Bond.

A3 Mrs. Bond ne possède aucun canard gris vivant dans ce village.

Traduisez ces affirmations sous la forme de règles Prolog dans un fichier `canards.pl` en vous appuyant sur les constantes `ce_village` et `mrs_bond`, les prédicats unaires `canard` et `porte_col_dentelle`, et les prédicats binaires
`appartient_a`, `marque_avec`, `pas_gris` et `vit_dans` (et aucun autre prédicat ou symbole de fonction).

**Q2.** À présent, on considère George, un canard de ce village marqué AB et qui porte un col de dentelles et Augusta, une cane de ce village marquée B. Ajoutez les faits correspondant à votre fichier.

**Q3.** On se demande quels sont les canards de la base qui ne sont pas gris. Utilisez l'interprète Prolog pour répondre à cette question.

---

**Exercice 4** L'atome `X is v` est évalué à `true` et a comme effet d'affecter à la variable `X` la valeur `v`. Cela permet par exemple de définir une suite en Prolog. Ainsi, la suite définie par

$$u_0 = 1 \qquad u_{n+1} = 2u_n \qquad \text{(pour tout entier naturel } n\text{)}$$

peut être implantée en Prolog de la façon suivante :

```
% Suite des puissances de 2
suite_u(0, 1).
suite_u(P, U) :-
    P >= 1,
    Q is P-1,
    suite_u(Q, V),
    U is 2 * V.
```

1. Le connecteur `not` correspond à une « négation par l'échec ». Par exemple, `not p(a, X)` donnera la valeur `true` si `p(a, X)` est non prouvable (cela a un lien avec l'hypothèse du monde clos vue en cours).

Pour que Prolog calcule $u_4$, il suffit de l'interroger avec `suite_u(4, X).` : la variable `X` sera unifiée à la valeur $u_4$.

Implantez et testez la suite de Fibonacci $(f_n)_n$ en Prolog. On rappelle que cette suite est définie ainsi :

$$f_0 = 1 \qquad f_1 = 1 \qquad f_{n+2} = f_{n+1} + f_n \qquad \text{(pour tout entier naturel } n\text{)}$$

———

**Exercice 5** Les listes en Prolog sont dénotées de la façon suivante `[a, b, c, ...]`. La liste vide est `[]` et si $L$ est une liste dont le premier élément est `P` et le reste `R`, on la note `[P | R]`. Les deux programmes Prolog suivant illustrent l'utilisation des listes en Prolog :

```
somme([], 0).
somme([P | R], X) :-
    somme(R, Y),
    X is P + Y.
```

```
supprimer(_, [], []).
supprimer(X, [X | R], L) :-
    supprimer(X, R, L).
supprimer(X, [Y | R], [Y | L]) :-
    X =\= Y,
    supprimer(X, R, L).
```

Le premier programme permet de calculer la somme des éléments d'une liste d'entiers : interrogé avec `somme([1, 2, 3, 4], S).`, l'interprète Prolog unifiera `S` à la valeur `10`.

Le deuxième programme permet permet de supprimer toutes les occurrences d'une valeur dans une liste : Ainsi si on interroge Prolog avec `supprimer(2, [1, 2, 3, 2, 4], X).`, la variable `X` sera unifiée avec la valeur `[1, 3, 4]`.

Dans le deuxième programme :

— La première règle (qui à une prémisse vide) a pour premier paramètre `_` : on utilise ce symbole comme une variable muette (si une variable n'apparaît qu'une fois dans une règle, il est inutile de la nommer et on la remplace par ce symbole) ;

— L'opérateur =\= teste que deux valeurs numériques sont différentes.

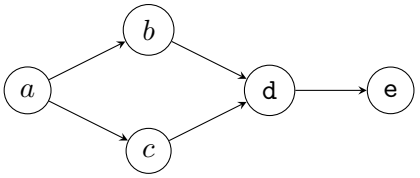
**Q1.** Donnez un programme Prolog qui permet de tester qu'une liste est triée de façon croissante. Le test `A =< B` permet de tester qu'un nombre `A` est inférieur ou égal à un nombre `B`.

**Q2.** Donnez un programme Prolog qui permette de renverser une liste. Ainsi `renverser([1, 2, 3, 4], L).` unifiera `L` à `[4, 3, 2, 1]`.

**Indication :** Vous pouvez faire cela en introduisant, en plus du prédicat binaire `renverser`, un prédicat ternaire `pa_renverser` (`pa` pour « prédicat auxiliaire »), tel que `pa_renverser(L, [], R)` entraîne `renverser(L, R)`.

———

**Exercice 6** On considère le graphe orienté sans circuit suivant :




**Q1.** En utilisant le prédicat binaire `arc`, représentez ce graphe par des faits en Prolog.

**Q2.** Comment faire afficher tous les arcs de ce graphe par une interrogation simple en Prolog (et en demandant le résultat suivant, tant qu'il y en a) ?

**Q3.** Comment afficher toutes les images du sommet `a` ?
Comment afficher tous les antécédents du sommet `d` ?

**Q4.** Donnez des règles pour le prédicat ternaire `chemin` tel que l'interrogation de `chemin(S1, S2, L)` unifie `L` avec un chemin de `S1` à `S2` (si un tel chemin existe).