LRC - TME n°5

1 Le langage *Prolog*

1.1 Syntaxe

Syntaxe des éléments de base

Prolog est un langage de programmation fondé sur la logique des prédicats du premier ordre [1].

- Les individus, qui correspondent aux constantes, sont représentés par des mots qui commencent par une minuscule.
- Les variables sont représentées par des mots qui commencent par une Majuscule.
- Les prédicats sont représentés par des mots qui commencent par une minuscule.

Ainsi, dans la proposition `frere(X,charlemagne)`, le prédicat est `frere`, d'arité 2 : il est appliqué à la variable `X` et à la constante `charlemagne`.

Syntaxe de la base de connaissances

Les règles sont de la forme `prop :- prop1, prop2, prop3.` qui correspond à la formule de LPPO $(prop1 \wedge prop2 \wedge prop3) \rightarrow p$.

Le prédicat à gauche de `:-` est la *tête* de la règle et les propositions à droite définissent le *corps* de la règle. Les propositions du corps sont séparées par des virgules.

Prolog exploite la représentation sous forme normale conjonctive des formules : ainsi, la formule $((prop_1 \wedge prop_2 \wedge prop_3) \vee (prop_4 \wedge prop_5)) \rightarrow prop$, dont la forme normale conjonctive est $(\neg prop_1 \vee \neg prop_2 \vee \neg prop_3 \vee prop) \wedge (\neg prop_4 \vee \neg prop_5 \vee prop)$, est représentée par les 2 règles

```
prop :- prop1, prop2, prop3.
prop :- prop4, prop5.
```

Chaque règle correspond à une clause de Horn, c'est-à-dire une clause comportant au plus un littéral positif.

Les faits correspondent aux clauses de Horn positives, qui contiennent un littéral positif et aucun littéral négatif. Ils sont représentés par des règles dont le corps est vide, c'est-à-dire de la forme : `prop.`

Stockage La base de connaissances, qui contient les faits et les règles, est écrite dans un fichier texte, dont le nom a pour extension `pl`. Il est préférable que le nom du fichier commence par une minuscule (cf ci-dessous la procédure de chargement du fichier).

Tout éditeur de texte peut être utilisé (l'éditeur `emacs` propose une coloration syntaxique pour prolog, si on utilise le mode prolog, en tapant alt+x prolog-mode).

Attention : il faut regrouper dans le fichier tous les faits et règles ayant la même tête, sinon l'interprète prolog provoque une erreur lors du chargement du fichier.

Les symboles `/* */` permettent de mettre des commentaires dans le fichier.

1.2 L'interprète `swiprolog`

L'interprète prolog sous Linux s'appelle `swipl`. Lorsqu'on lance le programme, on entre en mode "requête" indiqué par l'invite `?-`

Toutes les commandes se terminent par un point.

1. Les versions actuelles de prolog permettent de définir des formules d'ordre supérieur, mais nous n'aborderons pas ce point dans le cadre de LRC. Il existe aussi une variante modale de prolog.

- `[nom].` ou `consult(nom).` ou `['nom.pl'].` : charger le fichier `nom.pl`.
  La commande `[nom].` ne convient pas si le nom du fichier commence par une majuscule : prolog considère que l'on fait référence à une variable et provoque une erreur. Elle ne convient pas non plus si le nom du fichier contient des caractères spécifiques (tiret, point, etc).
  A chaque nouveau chargement d'un fichier existant, les faits et les règles précédents sont écrasés par les nouvelles définitions.
- `listing.` : visualiser l'état courant de la base de connaissances, c'est-à-dire savoir quels sont les faits et les règles que prolog a en mémoire.
- `help.` : afficher la fenêtre d'aide et la liste des prédicats prédéfinis dans SWI-Prolog.
- `help(pred).` : afficher l'aide concernant le prédicat `pred`.
- `trace.` : suivre pas à pas l'évaluation des requêtes.
- `notrace.` : quitter le mode trace.
- `halt.` : quitter Prolog.

1.3 Requêtes

Une fois le programme chargé, on peut entrer une requête, c'est-à-dire une proposition de la LPPO contenant des variables, se terminant par ".".

Considérons par exemple le cas où un fichier `toto.pl` contient les faits

```
toto(a,b).
toto(c,b).
```

Chargement du fichier

```
?- [toto].
//1 compiled 0.01 sec, 624 bytes
true.
```

Requête sans solution il n'existe pas de valeur du second argument de `toto` telle que le prédicat soit vérifié avec en premier argument `b`. Prolog répond `false`.

```
?- toto(b,X).
false.
```

Requête avec une ou plusieurs solutions prolog indique l'instanciation effectuée pour que le prédicat soit vérifié sous les conditions exprimées par la requête :

```
?- toto(a,X).
X = b.
```

S'il n'y a qu'une instanciation possible, l'interprète rend la main. Sinon, il attend une saisie de l'utilisateur :

```
?- toto(X,b).
X = a
```

La touche "Entrée" termine, la touche ";" donne successivement les autres solutions.

Tester les exemples précédents en mode trace, pour observer les étapes d'unification effectuées (après renommage des variables, de la forme $\_G2673$ ou un autre entier).

1.4 Tests d'égalité

Il existe en prolog plusieurs types d'égalité, faisant référence à des concepts différents. Ici, deux seulement sont rappelés, ceux qui portent sur des valeurs numériques sont omis.

- unifiabilité `=` et non-unifiabilité `\=` : la comparaison entre termes réussit s'il existe une instanciation des variables présentes qui rende égaux les deux termes. Ainsi

```
?- p(a,q(X)) = p(Y,q(b)).
X = b,
Y = a.
?- p(a,q(X)) = p(Y,r(b)).
false.
```

- identité `==` et non-identité `\==` : la comparaison entre termes réussit s'ils sont identiques ou s'ils sont liés à la même valeur :

```
?- X == a.
  false.
?- X = a.
   X = a.
?- X = 3, Y = 3, X == Y.
   X = Y, Y = 3.
?- X = 3, Y = 3, X \== Y.
   false.
```

2 Exercices

Exercice 1

On considère le programme formé des trois clauses suivantes :

```
r(a,b).
r(f(X),Y) :- p(X,Y).
p(f(X),Y) :- r(X,Y).
```

1* Calculer, à la main, les réponses aux requêtes `r(f(f(a)),b)` et `p(f(a),b)` en indiquant à chaque étape les inférences par résolution et les unifications réalisées.

2. Tester avec prolog, examiner et commenter la trace.

Exercice 2

On considère le programme formé des trois clauses suivantes :

```
r(a,b).
q(X,X).
q(X,Z) :- r(X,Y),q(Y,Z).
```

1. Calculer, à la main, les réponses à la requête `q(X,b)` puis à la requête `q(b,X)`.

2* Tester avec prolog, examiner et commenter la trace.

Exercice 3 Raisonnement simple

1. Donner la base de connaissances prolog représentant les assertions suivantes, en utilisant des règles et des faits

   Les étudiants sérieux révisent leurs examens. Un étudiant consciencieux fait toujours ses devoirs pour le lendemain. Les étudiants qui révisent leurs examens réussissent. Les étudiants qui font leurs devoirs pour le lendemain sont sérieux. Pascal et Zoé sont des étudiants consciencieux.

2. Donner la requête prolog qui permet de répondre à la question « qui va réussir ? ».
3. Calculer la réponse attendue à la main et vérifier que prolog donne le résultat attendu.
4. Dérouler l'exécution sous la forme du graphe de dérivation.

Exercice 4 Relations familiales*

Il est demandé de joindre au compte-rendu de TME le fichier `.pl` contenant les réponses aux questions suivantes, ainsi que des commentaires indiquant les tests réalisés et les résultats obtenus.

1. Définir quelques faits de type `pere` et `mere`, comme par exemple `pere(pepin, charlemagne)` (Pépin est le père de Charlemagne) ou `mere(berthe,charlemagne)`.

   N.B. Pour faciliter les vérifications des prédicats suivants, il est conseillé d'utiliser une famille que vous connaissez bien (typiquement la vôtre...).

2. Définir le prédicat `parent`/2 à partir des prédicats `pere`/2 et `mere`/2.

   Dans ces notations, l'entier après le slash indique l'arité des prédicats concernés : ici, les trois prédicats considérés ont deux arguments.

   `parent(X, Y)` doit être satisfait si `X` est le père ou la mère de `Y`.

   Ainsi, la requête `parent(X, charlemagne)` doit être satisfaite pour les deux instanciations `X = pepin` et `X = berthe`.

3. Tester (avec une base de faits plus conséquente) différents types de requêtes, comme par exemple `parent(charlemagne, X)`, `parent(pepin, Y)` ou `parent(A, B)`.
4. Définir et tester le prédicat `parents`/3, tel que `parents(X,Y,Z)` est satisfait si `X` est le père de `Z` et `Y` est la mère de `Z`.
5. Définir et tester les prédicats `grandPere`/2 et `frereOuSoeur`/2.
6. Définir et tester le prédicat `ancetre`/2 tel que `ancetre(X,Y)` est satisfait si `X` est un ancêtre de `Y`.

   Pour ce prédicat, faire varier la position des différentes propositions utilisées et commenter les résultats obtenus.

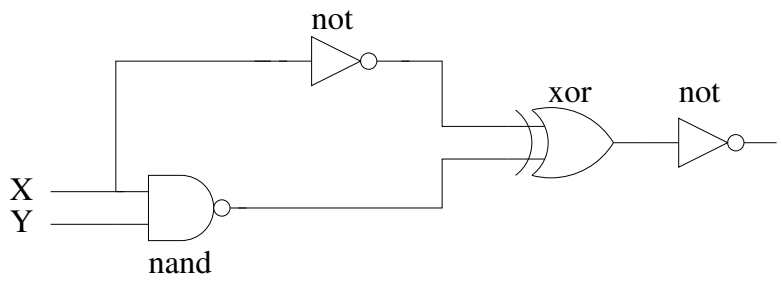
Exercice 5 Statuts entrée et sortie des variables*

Il est demandé de joindre au compte-rendu de TME le fichier `.pl` contenant les réponses aux questions suivantes, ainsi que des commentaires indiquant les tests réalisés et les résultats obtenus.

1. Définir les prédicats `et`/3, `ou`/3 et `non`/2 correspondant aux connecteurs logiques par un ensemble de **faits**, en utilisant les constantes 0 et 1 pour les entrée et sortie de chaque connecteur.

   Ainsi `et(0,1,0).` définit que le prédicat `et` est vérifié pour le triplet (0,1,0), ce qui représente le fait que le et logique de 0 et 1 vaut 0. 4 faits de ce type définissent donc le prédicat complètement.

   Particularité importante de prolog Il faut noter que prolog ne définit pas des fonctions qui prennent des arguments et *renvoient* des résultats, contrairement aux langages de programmation classique : il énonce le fait qu'*il est vrai qu'il existe une relation*, définie par le nom de la fonction, entre les paramètres d'entrée et de sortie.
2. Tester différents types de requêtes de la forme `et(X,Y,1)`, `et(0,0,R)` ou `et(X,Y,R)`.
3. Définir le prédicat `circuit`/3 tel que `circuit(X, Y, Z)` est satisfait si et seulement si `X` et `Y` correspondent à des valeurs en entrée pour lesquelles `Z` est la valeur de sortie du circuit indiqué sur la figure ci-dessous.



4. Donner la requête permettant de construire la table de vérité du circuit.

   NB la table de vérité attendue est celle de l'implication.
5. Tester différents types de requêtes en faisant varier le statut variable/valeur fixée de chacun des arguments.