Université de Reims
Champagne-Ardenne
U.F.R. de Sciences
Exactes et Naturelles

Licence 3 INFO / PASSERELLE
INFO0502
2019/2020
J.-C. Boisson

TP 1

Premiers pas avec Prolog

1 Faits, Règles et Requêtes

La programmation en Prolog est **déclarative** : cela signifie que l'on ne décrit pas un algorithme qui résout un problème, mais plutôt les données et les relations qui décrivent le problème lui-même. Les données sont décrites dans une base de **faits** et les relations dans une base de **règles**. La résolution elle-même est réalisée par l'interpréteur Prolog lorsque l'on fait une requête i.e. lorsqu'on lui pose une question.

Les faits sont considérés comme des axiomes c'est à dire des hypothèses considérées comme vraies. Un fait peut être vu comme la simple déclaration d'un fonction avec ou sans paramètre(s) le tout écrit en minuscule et surtout finissant par un point. Voici un exemple de fait sans paramètre :

$$vivant.$$

Dans la base associée, si on demande (le " ?-" représente le prompt de l'interpréteur Prolog) :

$$?- vivant.$$

La réponse sera :

$$true.$$

Voici maintenant un exemple de fait avec des paramètres (en minuscule c'est à dire constant) :

$$est\_contente(lily).$$

Dans la base associée, si on demande :

$$?- est\_contente(lily).$$

La réponse sera :

$$true.$$

Si on cherche à savoir qui "est_content", il faut utiliser une variable c'est à dire une nom commençant par une majuscule.

Si on demande :

$$? - est\_contente(Qui).$$

La réponse sera :

$$Qui = lily.$$

Voici un exemple plus complexe avec plusieurs faits paramétrés :

```
assure_au_paint_ball(arnaud).
assure_au_paint_ball(christophe).
a_battu_au_paint_ball(arnaud, cyril).
a_battu_au_paint_ball(christophe, cyril).
a_battu_au_paint_ball(florent, cyril).
a_battu_au_paint_ball(olivier, cyril).
```

Une fois les faits établis, on peut effectuer des requêtes dans l'interpréteur :

Requête	*Réponse de l'interpréteur*
?- assure_au_paint_ball(R).	2 solutions : {R = arnaud} et {R = christophe}
?- a_battu_au_paint_ball(arnaud, X).	1 solution : {X = cyril}
?- a_battu_au_paint_ball(cyril, Y).	l'interpréteur répond No : il n'y a aucune solution
?- assure_au_paint_ball(arnaud).	l'interpréteur répond Yes : c'est juste
?- assure_au_paint_ball(X), a_battu_au_paint_ball(X, cyril).	2 solutions : {X = arnaud} et {X = christophe}
?- a_battu_au_paint_ball(W, Z), assure_au_paint_ball(W).	2 solutions : {W = arnaud, Z = cyril} et {W = christophe, Z = cyril}

On notera que la virgule est utilisée pour décrire un but plus évolué, elle est équivalent au "et".

2 Les données

En Prolog toutes les donnés sont des termes. Il y a 4 types de termes : les atomes, les nombres, les variables et les termes complexes (prédicats). Les atomes représentent la famille des chaînes de caractères commençant par une minuscule ou qui sont entourées par des simples quotes. Les variables ont des noms commençant forcément par une majuscule. Si ces dernières ne sont pas encore instanciées (unifiées), on les qualifie de libres. Les prédicats sont quand à eux formés d'un nom suivi de parenthèses qui comportent en leur sein un ou plusieurs arguments.

3 Utilisation de SWI Prolog

La version actuelle de L'interpréteur utilisé lors de la rédaction de ces TP est `SWI-Prolog` (`8.0.3`) (compilateur et interpréteur). SWI-Prolog est maintenu par l'Université de Psychologie d'Amsterdam et est téléchargeable gratuitement (licence GPL) *http ://www.swi-prolog.org/*. Il est disponible sous Unix, MacOS et Windows. Il est aussi généralement proposé dans les dépôts de la plupart des distributions Linux (il peut être appelé ppl-swiprolog). A défaut, vous pouvez utiliser GProlog.

Un programme Prolog (contenant les faits) est décrit dans un fichier source d'extension `.pl` ou encore `.prolog` pour ne pas le confondre avec du Perl suivant votre éditeur. On peut tester un programme soit en l'interprétant interactivement, soit en le compilant au préalable, soit en l'interfaçant avec du c, c++ ou java.

— *Solution compilée* : je crée mon programme (xxx.prolog) et je le compile par `swipl -o xxx -c xxx.prolog` (-c en dernier !). Puis j'appelle `xxx`, il me propose de donner des buts (je dois les terminer par .), je quitte par halt, (le point compris !) ou CTRL+D ;
— *Solution interactive* : Le programme s'utilise dans l'interpréteur Prolog, qu'on lance avec la commande swipl.

Dans l'interpréteur, le fichier est chargé par **consult('fichier.prolog').** ou par le raccourci équivalent **['fichier.prolog'].** (**Ne pas oublier le point**, l'extension de fichier peut être ignorée).

Il est ensuite possible d'effectuer une requête dans l'interpréteur (par exemple **requete(X).**) Prolog affiche la première solution qu'il trouve (par exemple $X = 3$). Pour voir la prochaine solution, taper **;**. Pour terminer (c'est-à-dire accepter la solution), taper <Entrée>.

3.1 Quelques instructions

— **trace** : pour afficher l'arbre de déduction, on utilise le mode trace dans lequel pour chaque pas on appuie sur la touche <Entrée> (h pour les autres options) :

$$trace, ma\_question(MesParametres).$$

— **listing** : affiche tout ce qu'il y a dans la base (notamment les faites/règles par défaut du système) à l'écran. **listing(predicat).** permet de se limiter à un seul prédicat.

— Pour les chaînes de caractères, les mettre entre simples cotes (elles seront affichées par **write** et **writef**). Entre doubles cotes elles sont transformées en listes d'entiers (d'après les codes ascii). Elles seront affichées en liste d'entiers par write mais en chaîne de caractères par writef. Entre guillemets elles peuvent être traitées comme toute autre liste.

— **findall** : pour afficher toutes les solutions, au lieu de taper **;** après chaque réponse, on utilise **findall : findall(X,a_battu_au_paint_ball(X,cyril), ListeDeTousLesJoueurs).** qui crée la liste de toutes les unifications de X vérifiant a_battu_au_paint_ball(X) (X et ListeDeTousLesJoueurs sont normalement libres).

— **_ (underscore)** désigne une variable anonyme qu'on n'utilisera pas explicitement par son nom. Par exemple, l'ajout de la règle a_battu_au_paint_ball(_,cyril) permet de dire que tout le monde a battu cyril.

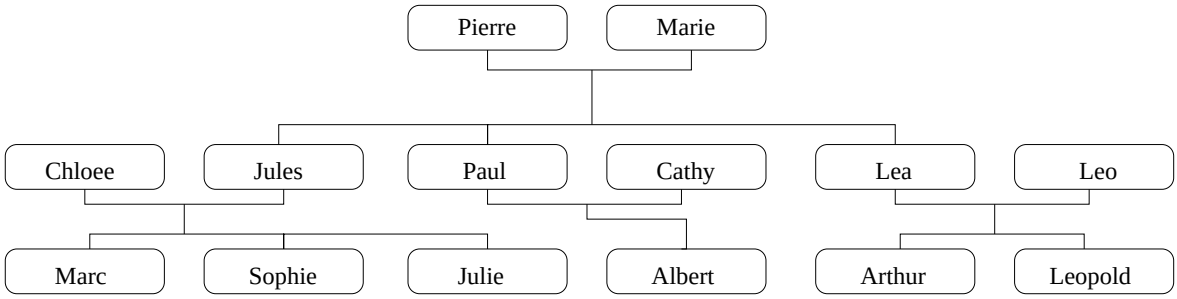
Pour l'aide (doc en ligne) :

— help. : racine de l'aide
— help(1). : introduction générale
— help(2). : options de ligne de commande, historique, différents modes, ...
— help(3). : gérer un projet en Prolog
— help(4). : les prédicats par défaut
— help(5). : nouveautés de la version 7
— help(6). : les modules
— help(7). : spécificités bas niveau pour la programmation par contrainte
— ...

3.2 Informations sur Prolog

— Site SWI Prolog : http ://www.swi-prolog.org
— Site très complet (en anglais) : http ://www.learnprolognow.org

4 Exemple : Arbre généalogique



4.1 Description

Décrire l'arbre généalogique ci-dessus à l'aide des prédicats suivants :
— *male(X)* pour dire que X est de sexe masculin
— *femelle(X)* pour dire que X est de sexe féminin
— *enfant(X, Y)* pour dire que X est enfant de Y
— *mari(X, Y)* pour dire que X est le mari de Y
Ecrivez l'ensemble de faits pour décrire l'arbre généalogique

4.2 Tests

Effectuez quelques tests pour vérifier que la liste de faits saisie est correcte (vous utiliserez l'arbre de déduction pour analyser les réponses proposées par l'interpréteur) :

1. Quels sont les enfants de Jules
2. Qui est la femme de Paul
3. Quels sont les enfants mâles de Marie
4. Quels sont les couples père/fille
5. Quels sont les couples frère/soeur
6. Quelles sont les petites-filles de Pierre
7. Quels sont les oncles de Sophie
8. Quels sont les beaux frères de Chloée

5 Les règles

De la même manière que l'on a défini les faits, on peut définir des règles qui utilisent la base de faits. Par exemple, pour savoir si X est le père de Y, on définira une règle qui comporte une seule clause :

$$pere(X, Y) : -enfant(Y, X), male(X).$$

Par exemple, pour savoir si X est le grand-père de Y, on définira une règle qui comporte deux clauses :

$$grandpere(X, Y) : -pere(X, Z), pere(Z, Y).$$
$$grandpere(X, Y) : -pere(X, Z), mere(Z, Y).$$

Une fois ces règles ajoutées à vos faits, vous pouvez effectuer des requêtes dans l'interpréteur (notez l'arbre de déduction) :

Requête	*Réponse de l'interpréteur*
?- grandpere(B, arthur).	1 solution : {B = pierre}
?- grandpere(X,Y).	6 solutions :{X = pierre,Y = marc} et {X = pierre,Y = sophie}, ...
?- grandpere(pierre,marc).	l'interpréteur répond Yes : c'est juste.
?- grandpere(pierre,leo).	l'interpréteur répond No : il n'y a aucune solution.

A chaque fois qu'une règle a une même partie gauche, avant le " :-", l'intépréteur fera un "ou" entre les différentes parties droites dans l'ordre de déclaration.

6 Résolution

Supposons qu'on ait donné les faits et la règle suivante :

```
f(a).
f(b).
g(a).
g(b).
h(b).

k(X) :- f(X), g(X), h(X).
```

Lorsqu'on fait la requête k(X), Prolog cherche une valeur pour X telle que f(X), g(X) et h(X) sont (tous) vrais. Dans la pratique, Prolog cherche d'abord le premier X tel que f(X) : il trouve a, et cette solution est aussi correcte pour g(X). Mais si X=a, h(X) est faux, Prolog doit chercher une autre solution par *backtracking* (retour en arrière).

- Dans l'interpréteur, la commande trace, permet de voir pas à pas comment Prolog résout un appel. Vérifiez avec l'exemple ci-dessus.
- L'ordre des règles a de l'importance car Prolog les utilise dans l'ordre ou elles sont définies. Le mode trace permet aussi de mettre en évidence ce mécanisme.

— Le prédicat *fail* est toujours faux. Il sert à obliger Prolog à chercher une autre solution par backtracking.
— A l'inverse, le prédicat *true* est toujours vrai.
— La virgule , entre deux conditions C1 et C2 d'une clause-indique qu'il faut que C1 ET C2 soient vraies.
— Le point-virgule ; entre deux conditions indique qu'il faut que C1 OU C2 soit vraie. Quand la règle comporte un grand nombre de lignes on préférera ne pas utiliser le ;

7 Exemple : Arbre généalogique (suite et fin)

A la suite du travail effectué précédemment, écrire les prédicats suivants :

1. individus(X) (indique indifféremment les mâles et les femelles)
2. mère (X,Y) et grandmere(X,Y)
3. frere(X,Y) et soeur(X,Y)
4. femme(X,Y) (X est la femme de Y)
5. oncle(X,Y), tante(X,Y), beaufrere(X,Y), bellesoeur(X,Y)
6. cousin(X,Y)
7. gendre(X,Y), brue(X,Y), beaupere(X,Y), bellemere(X,Y)
8. grandparent(X,Y), ancetre(X,Y) (X est un ancêtre de Y)
9. famille(X,Y) (X et Y sont de la même famille, ajouter d'autres faits pour vérifier)

Posez des questions au système (ex : ?- soeur(Marie,Arthur) ou ?-frere(X,Marie)) et noter l'arbre de déduction.