Initiation à la Programmation en Logique
avec
SICStus Prolog

Identificateurs

Ils sont représentés par une suite de caractères alphanumériques commençant par une lettre minuscule (les lettres accentuées sont à éviter).

Exemples : f, aLPHA, jean-paul, h1.

Variables

Elles sont représentées par une suite de caractères alphanumériques commençant par une lettre majuscule (les lettres accentuées sont à éviter).

Exemples : X, X1, MACHIN, Truc.

Prédicats (et fonctions)

Ils sont représentés à l'aide d'un identificateur. Les arguments sont écrits après, entre parenthèses, séparés par des virgules (notation préfixée et parenthésée). Le nombre d'arguments d'un prédicat définit son poids.

Constantes

Elles sont représentées avec un nombre ou un identificateur.

Clauses

Une clause Prolog est définie par une tête et éventuellement par un corps (une clause Prolog ne possède pas forcément un corps).

La tête de clause est formée par un seul prédicat, tandis que le corps est constitué de un ou plusieurs prédicats ou formules qui sont séparés par des virgules.

La tête est séparée du corps par les signes ' :-'. Une clause Prolog se termine toujours par un point.

Soit une formule F définie sous forme normale disjonctive (DNF) par :
$$F = (A_1^1 \wedge A_2^1 \wedge ... \wedge A_{n_1}^1) \vee (A_1^2 \wedge A_2^2 \wedge ... \wedge A_{n_2}^2) \vee ... \vee (A_1^m \wedge A_2^m \wedge ... \wedge A_{n_m}^m)$$

Cette formule F sera représentée en Prolog par le prédicat `f` tel que :
f :- $\mathtt{A}_1^1, A_2^1, ..., A_{n_1}^1$.
f :- $\mathtt{A}_1^2, A_2^2, ..., A_{n_2}^2$.
f :- $\mathtt{A}_1^m, A_2^m, ..., A_{n_m}^m$.

Ainsi, les conjonctions sont remplacées dans les clauses Prolog par des virgules et chaque ligne débutant par une même tête de clause est séparée par une disjonction. Ainsi, dans le cadre d'une unification, les lignes sont testées les unes après les autres dans leur ordre d'apparition dans le fichier.

Exemple de programme

monFichier.pl

```
% Exemple 1 : cette ligne est un commentaire
habite(frederic,vesoul).
habite(fabrice,marseille).
habite(fabien,belfort).
habite(jacques,vesoul).

meme_endroit(X,Y) :-
        habite(X,V),
        habite(Y,V).
```

Pour pouvoir poser des questions à ce petit exemple, il faut dans un premier temps le saisir dans un éditeur de texte (Emacs par exemple) et l'enregistrer au format texte (les fichiers Prolog ont habituellement l'extension .pl).

Dans un deuxième temps, il faut lancer SICStus Prolog. Les symboles '`| ?-`' indique que le programme est en attente d'une question. Avant de poser des questions, il est nécessaire de mettre dans la mémoire de SICStus les clauses (faits et règles) enregistrées dans le fichier. Cette opération est effectuée au moyen de la commande `consult` de la manière suivante :

```
| ?- consult('Z:\\mon_repertoire\\...\\monFichier.pl').
```

Si la consultation du fichier se déroule correctement, la console Prolog répond `yes` : on peut dès lors questionner le programme. Sinon, les erreurs détectées dans le programme

sont présentées et doivent être corrigées dans le fichier source `monFichier.pl`.

Exemples de questions

On ne pose à Prolog qu'une question à la fois. Une question prend la forme d'un unique prédicat (ou d'une conjonction de prédicats séparés par des virgules). Ne jamais oublier le point qui doit terminer toute question.

Si la question ne contient aucune variable, Prolog répond `yes` ou `no` suivant que les règles et faits (contenus dans le fichier préalablement consulté) permettent ou non de déduire le prédicat de la question :

```
| ?- habite(fabrice,marseille).
yes

| ?- habite(fabrice,vesoul).
no

| ?- habite(fabrice,marignane).
no
```

Si la question contient au moins une variable, Prolog retourne une à une les configurations de valeurs qui rendent la question posée vraie. Plus exactement, il retourne une synthèse des unifications posées lors du parcours de l'arbre de résolution.
Ainsi, à une telle question, Prolog retourne le premier résultat suivi d'un point d'interrogation. En frappant un point virgule, on obtiendra le second résultat, et ainsi de suite. En frappant, un retour chariot, on arrête le parcours de l'arbre de résolution. La réponse `no` précise qu'il n'y a pas ou plus de solution, tandis que `yes` signifie que l'arbre n'a pas été parcouru entièrement (dans le cas d'un retour chariot).

```
| ?- habite(fabien,X).
X = belfort ? ;
no

| ?- habite(X,vesoul).
X = frederic ?
yes

| ?- meme_endroit(X,jacques).
X = frederic ? ;
X = jacques ? ;
no
```

A noter que chaque argument d'un prédicat peut indifféremment jouer le rôle de paramètre d'entrée (si c'est une constante) ou de paramètre de sortie (si c'est une variable).

Tracer un programme Prolog

Il est possible de suivre pas à pas l'exécution d'un programme Prolog. Pour cela, il suffit de taper `trace.` dans la console Prolog avant de poser des questions.
Cette routine de trace se désactive par la saisie de `notrace.` dans la console Prolog lorsque Prolog est en attente d'une question. Sinon, elle peut être abandonnée pour la question courante par la saisie, en fin de ligne, de la lettre `n` (signifiant `notrace`). D'autres options sont disponibles et consultables par la saisie de la lettre `h` (pour `help`).

Listes

Les listes sont décrites à l'intérieur de crochets. Pour former des listes, on utilise en plus des crochets l'un ou l'autre symbole : `,` ou `|`. Ces symboles sont des séparateurs, mais attention, leur signification n'est pas du tout la même.

La virgule est un séparateur permettant d'énumérer les éléments d'une liste. Ainsi, `[a,b,c]` est la liste formée par les éléments `a`, `b` et `c`. La liste `[]` désigne la liste vide.

`L = [E|L1]` définit que la liste `L` doit être unifiée à la liste obtenue en ajoutant l'élément `E` au début de la liste `L1`. On a donc `L = [Tête|Queue]` où `Tête` représente le premier élément de la liste `L`, tandis que `Queue` représente le reste de la liste `L`.
Ainsi, ce symbole permet de récupérer le premier élément d'une liste (lorsque la liste `L` est connue), ou d'ajouter un élément en début de liste (lorsque `E` et `L1` sont connus).

Les notations suivantes désignent ainsi la même liste : `[a,b]`, `[a|[b]]`, `[a|[b|[]]]`.

Exemples de listes :
```
[1,2,3,4],
[],
[[il,fait,beau],[jean,se,baigne]],
[1,B,1].
[[1,voiture],X,3].
```

Exemples d'unifications : `[T|Q]` et `[vert,orange,rouge]` s'unifient en :
`T=vert` et `Q=[orange,rouge]`.

`[T,Q]` et `[vert,orange,rouge]` ne s'unifient pas.

`[T1,T2|Q]` et `[vert,orange,rouge]` s'unifient en :
`T1=vert`, `T2=orange` et `Q=[rouge]`.

`[T1,T2,T3|Q]` et `[vert,orange,rouge]` s'unifient en :
`T1=vert`, `T2=orange`, `T3=rouge` et `Q=[]`.

Récursivité

Simuler pas à pas le traitement de questions aux deux exemples de prédicats donnés ci-dessous (`appartient/2` et `concatene/3`) et comprendre la construction de leur(s) résultat(s). Ces deux exemples doivent absolument être maitrisés avant de s'engager dans une programmation plus avancée.

monFichier.pl

```
% Exemple 2 : appartient(X,L) ⇔ 'X appartient à L'
appartient(X,[X|Y]).
appartient(X,[Z|Y]):-
          appartient(X,Y).
```

Deux procédés peuvent être envisagés pour implanter le prédicat `concatene/3` : soit utiliser un accumulateur (le résultat est construit lors de l'empilement des appels de prédicats) ou ne pas en utiliser (le résultat est construit lors du dépilement des appels de prédicats).

monFichier.pl

```
% Exemple 3 : concatene(L1,L2,R) ⇔ 'R est la concaténée de L1 et L2'

% Approche avec accumulateur (solution 1)
concatene1([],L,L).
concatene1([X|L1],L2,R):-
          concatene1(L1,[X|L2],R).

% Approche sans accumulateur (solution 2a)
concatene2a([],L,L).
concatene2a([X|L1],L2,R):-
          concatene2a(L1,L2,R1),
          R=[X|R1].

% Approche sans accumulateur (solution 2b)
concatene2b([],L,L).
concatene2b([X|L1],L2,[X|R1]):-
          concatene2b(L1,L2,R1).
```

Exemples de questions :

```
|?- appartient(b,[a,b,c]).          |?- appartient(t,[a,b,c]).
|?- appartient(X,[a,b,c]).          |?- appartient(a,X).
|?- concatene([a,b],[c],[a,b,c,d]). |?- concatene([a,b],[c,d],L).
|?- concatene(L,[2,3],[1,2,3]).     |?- concatene(L1,L2,[1,2,3]).
```

Nombres

Les nombres sont des constantes. Ils peuvent faire l'objet d'opérations notées de façon infixée avec +, -, *, / , // (division entière), mod. Ces opérations sont utilisées avec le prédicat `is`. Les conditions, dans lesquelles on peut utiliser ce prédicat, sont assez strictes.

Ainsi, on doit écrire à gauche de `is` un nombre ou une variable. A sa droite, on doit trouver une expression numérique. Cette expression peut contenir des variables à condition que ces dernières aient fait l'objet d'une unification avec un nombre avant le traitement de la clause contenant `is` (une sorte d'initialisation des variables du membre droit). Ainsi, la question `X is Y+3.` n'est pas acceptée mais la question `Y is 0, X is Y+3.` est valable.

Le traitement de `is` consiste en une unification. SICStus va effectivement tenter d'unifier le membre de gauche avec le résultat du membre droit. Si l'unification échoue, la valeur `no` est renvoyée. Si elle aboutit, l'unification est effectuée (le membre gauche prend la valeur du résultat du membre droit) et la valeur `yes` est renvoyée.

Il est possible de comparer des nombres au moyen de `<,=<,...` La notation est infixée, et les expressions à droite et à gauche du prédicat de comparaison peuvent également contenir des variables si celles-ci sont instanciées par unification à un nombre avant le traitement de la clause de comparaison.

Prolog propose également quelques fonctions sur les entiers, comme les fonctions `min` et `max` qui sont d'arité 2. Ces deux fonctions permettent respectivement de retourner la plus petite valeur et la plus grande valeur de deux nombres. Ainsi, la valeur de `min(X,Y)` est la plus petite valeur de X et de Y, tandis que `max(X,Y)` correspond à la plus grande.

Not (\+)

Le 'not' permet de considérer la négation d'un prédicat. Ainsi, soit l'exemple suivant dans lequel la clause `complement(X,L,M)` définit 'X appartient à la liste L et n'appartient pas à la liste M'.

monFichier.pl

```
% Exemple 4 : utilisation de \+
complement(X,L,M):-
          appartient(X,L),
          \+ appartient(X,M).
```

Utiliser cette construction avec précaution car elle peut induire des effets de bord mettant en échec le processus de résolution. Pour éviter de tels effets, on appliquera toujours l'opérateur 'not' sur un unique prédicat dont tous les arguments sont valués lors de l'appel.

Lecture et écriture à l'écran

Pour réaliser de la lecture et de l'écriture à l'écran, on utilise les prédicats de poids 1 `read` (lecture) et `write` (écriture).

monFichier.pl

```
% Exemple 5 : utilisation de read/1 et write/1
double:-
          write('Entrez un nombre :'),
          read(A),
          Z is 2*A,
          nl,
          write('Le double de '),
          write(A),
          write(' est '),
          write(Z).
```

Pour faire fonctionner cet exemple, questionner avec : `|?- double.` (ne pas oublier le point à la fin de chaque entrée). Noter que le prédicat de poids 0, `nl` permet de passer à une nouvelle ligne.

Cut (!)

Ce mécanisme de contrôle du parcours de l'arbre est très important. Il est noté `'!'`. Il peut être placé dans toute clause, comme un prédicat prédéfini de poids 0. Lors du parcours de l'arbre, la clause le contenant ne rendra plus qu'un seul résultat après avoir franchi le cut. Pour cette raison, on nomme ce prédicat un coupe-choix.

Revenons à l'exemple 2 qui définissait `appartient`, et ajoutons un cut :

monFichier.pl

```
% Exemple 6 : appartient_1(X,L) ⇔ 'X appartient à L'
appartient_1(X,[X|_]):-
          !.
appartient_1(X,[_|Y]):-
           appartient_1(X,Y).
```

Comparer le résultat de `appartient(X,[a,b,c]).` et de `appartient_1(X,[a,b,c]).`, et suivre le déroulement pas à pas de `appartient_1` afin de comprendre le mécanisme du cut.

Exemples récapitulatifs

On veut construire un prédicat `pair/2` qui permet de renvoyer la liste des nombres pairs d'une liste L de nombres donnée en argument. Rappelons qu'un nombre est pair si le reste de sa division par 2 est égal à 0. On va ainsi appliquer modulo 2 sur chacun des termes de la liste `L` pour tester s'il est pair. Si c'est le cas, il doit être présent dans la liste résultat.

monFichier.pl

```
% Exemple 7a : Approche avec accumulateur (solution 1)
pair1(L,L_Nombres_Pairs):-
          pair1(L,[],L_Nombres_Pairs).

pair1([],Acc,Acc).
pair1([E|L],Acc,L_Nombres_Pairs):-
          0 is E mod 2,
          pair1(L,[E|Acc],L_Nombres_Pairs).
pair1([E|L],Acc,L_Nombres_Pairs):-
          R is E mod 2,
          R\=0,  % ou: dif(R,0),
          pair1(L,Acc,L_Nombres_Pairs).

% ------------------------------------------
% Exemple 7b : Approche avec accumulateur et cut (solution 2)
pair2(L,L_Nombres_Pairs):-
          pair2(L,[],L_Nombres_Pairs).

pair2([],Acc,Acc).
pair2([E|L],Acc,L_Nombres_Pairs):-
          0 is E mod 2,
          !,
          pair2(L,[E|Acc],L_Nombres_Pairs).
pair2([_|L],Acc,L_Nombres_Pairs):-
          pair2(L,Acc,L_Nombres_Pairs).

% ------------------------------------------
% Exemple 7c : Approche sans accumulateur et cut (solution 3)
pair3([],[]).
pair3([E|L],[E|L_Nombres_Pairs]):-
          0 is E mod 2,
          !,
          pair3(L,L_Nombres_Pairs).
pair3([_|L],L_Nombres_Pairs):-
          pair3(L,L_Nombres_Pairs).
```