Découvrir Prolog en quelques exemples

Exemple 1

Voici un premier « Programme » Prolog :

```
riche(picsou).
riche(flairsou).
riche(gripsou).
```

Un tel programme ne s'exécute pas dans le sens habituel du terme. On lui soumet un but que Prolog essaie de satisfaire.

```
> riche(flairsou).
Réponse : Yes.

> riche(donald).
Réponse : no.
```

> Comment Prolog sait-il que Donald n'est pas riche ?

```
> riche(X).
Réponse : X/picsou, X/flairsou, X/gripsou.
```

Prolog recherche toutes les manières de satisfaire le but .

Exemple 2

```
riche(picsou). riche(flairsou). riche(gripsou).
oncle(picsou,donald). oncle(donald,riri).
oncle(donald,fifi). oncle(donald,loulou).
```

Le but suivant permet de rechercher les entités Y ayant un oncle riche :

```
> oncle(X,Y),riche(X).
Réponse : X/picsou Y/donald.
```

> Quel est le résultat du but : `riche(A), oncle(A,B), riche(B).` Donnez en une « traduction » en français.

Exemple 3

```
riche(picsou). riche(flairsou). riche(gripsou).
chanceux(gontran). chanceux(picsou).
jeune(flairsou). jeune(donald).

p(X) :- chanceux(X),jeune(X).
p(X) :- riche(X).
```

Le prédicat p est spécifié par deux clauses. Chacune d'elles exprime une manière de satisfaire p. Donc p(X) est vrai si X est jeune et chanceux ou si X est riche.

Donnez le résultat de `p(A)`.

Calculer avec Prolog

Exemple 4

```
carre1(X,Y) :- Y is X*X.
carre2(X,Y) :- Y=X*X.
carre3(X,Y) :- Y==X*X.

> carre1(3,Y)
Réponse : Y=9.

> carre1(X,9).
Réponse : Error.
```

L'opérateur `is` permet de réaliser un calcul à condition que toutes les valeurs apparaissant à gauche soient connues au moment de l'évaluation. Il peut aussi comparer une valeur connue et le résultat d'un calcul :

```
> carre(3,9).
Réponse : Yes.

> carre2(3,R).
Réponse :  R / '*'(3,3).
```

L'opérateur = réalise une *unification* entre les termes de gauche et de droite, i.e., il essaie d'assigner aux variables des valeurs qui rendent ces termes identiques. Mais il ne réalise pas de calcul. 3*3 est interprété comme une notation infixe du terme fonctionnel *(3,3).

> Quel est le résultat du but `carre2(X,R).` ?

```
carre3(X,3).
Réponse : no.
```

L'opérateur == réalise une comparaison syntaxique sans tenter d'unifier les termes. Syntaxiquement, `*(3,3)` est différent de 3, donc le but échoue.

Exemple 5

```
ticket(Age,5) :- Age<10.
ticket(Age,8) :- Age<16.
ticket(Age,10) :- Age>=16.
```

Le prédicat `ticket` permet de calculer le prix d'un ticket d'entrée à une attraction en fonction de l'âge de la personne concernée. Il est spécifié par trois clauses. La première permet de satisfaire `ticket(Age,5)` lorsque la condition (sous-but) `Age<10` est réalisée. Si cette condition est fausse, la clause échoue mais Prolog tentera de satisfaire le but avec la deuxième clause, et ainsi de suite.

```
ticket(14,P).
Réponse : P / 8.
```

Dans l'exemple précédent, Le but demande le prix du ticket pour une personne de 14 ans. C'est la deuxième clause qui permet de fournir un résultat. Mais il y a un problème avec le but suivant :

```
ticket(5,P).
Réponse : P/5, P/8.
```

En effet, Prolog explore *toutes* les possibilités de satisfaire le but. Dans le cas d'un enfant de 5 ans, le but peut être satisfait par les deux premières clauses et donc il y a deux résultats.

Modifiez la deuxième clause de manière à éviter ce problème.

Il existe un autre moyen que celui qui vient d'être utilisé. Il consiste à utiliser une instruction appelée « coupe-choix » et représentée par un point d'exclamation (!), qui efface tous les choix en attente.
Le programme s'écrit alors :

```
ticket(Age,5) :- Age<10,!.
ticket(Age,8) :- Age<16.
ticket(Age,10) :- Age>=16.
```

Et peut même être encore simplifié :

```
ticket(Age,5) :- Age<10,!.
ticket(Age,8) :- Age<16,!.
ticket(_,10).
```

Le caractère de soulignement indique que la valeur du premier slot du prédicat n'intervient pas. Si les deux premières clauses ont échoué, le prix du ticket est de 10 Euros.

Ce coupe-choix est très pratique mais il érode fortement la déclarativité de Prolog. La dernière variante du programme pourrait se traduire par : « Si Age < 10 alors le prix est de 5 Euros, sinon si Age < 16 le prix est de 8 Euros sinon c'est 10 Euros. »

Listes en Prolog

Toute liste est soit une liste vide notée [], soit une liste non vide constituée d'un élément de tête et d'une queue qui est une liste. Dans ce cas, elle peut être unifiée avec l'expression [T|Q].

Exemple 6

Voici comment on peut spécifier un prédicat permettant de vérifier si un élément appartient à une liste :

```
appartient(X,[X|_]).
appartient(X,[_|Q]) :- appartient(X,Q).
```

Ce prédicat est spécifié par deux clauses et il y a donc deux moyens de le satisfaire : en ayant dans le deuxième slot une liste commençant par la valeur du premier slot, ou en ayant dans le deuxième slot une liste dont la queue contient X. Cette deuxième condition est exprimée par une spécification récursive de la deuxième clause.

```
> appartient(3,[1,3,5]).
Réponse : yes.
```

Ce prédicat peut aussi être utilisé pour produire tous les éléments de la liste :

```
> appartient(T,[1,3,5]).
Réponse : T / 1, T / 3, T /2.
```

Autrement dit, le premier slot peut être utilisé comme entrée (avec une valeur connue) ou comme sortie (avec une variable non instanciée). Cette réversibilité a toutefois des limites que nous aurons l'occasion d'explorer au cas par cas.

Donnez la spécification d'un prédicat `dernier/2` tel que si L est une liste connue alors le but `dernier(L,T)` instancie T avec le dernier élément de L.
Par exemple, le but `dernier([1,2,4],R)` doit produire le résultat `R=4`.

Exemple 7

```
filtre([],[]):-!.
filtre([T|Q],[T|R]):- 0 is mod(T,2),!,filtre(Q,R).
filtre([T|Q],R):-filtre(Q,R).
```

En examinant les trois clauses ci-dessus, tentez de déterminer l'utilité du prédicat filtre et le résultat du but `filtre([2,3,3,4,4,1],L).`

Arbres en Prolog

Les arbres peuvent être représentés par des termes fonctionnels. Par exemple `n(12,n(11,f,f),f)` peut, par convention, représenter un arbre binaire étiqueté par des entiers avec 12 à la racine, un fils gauche ayant 11 à sa racine et deux feuilles, et un fils droit limité à une feuille. Les symboles fonctionnels n et f sont arbitraires et il n'est pas nécessaire de déclarer un type comme c'est le cas avec les variantes en CAML.

Exemple 8

Le prédicat suivant permet de rechercher si une valeur est dans un arbre représenté avec la convention ci-dessus :

```
isIn(T,n(T,_,_)):-!.
isIn(T,n(_,G,_)) :- isIn(T,G),!.
isIn(T,n(_,_,D)) :- isIn(T,D).
```

Ce prédicat est spécifié par trois clauses.

- La première permet de satisfaire le prédicat dans le cas où la valeur recherchée est à la racine.
- La deuxième permet de le satisfaire si la valeur recherchée est dans le fils gauche.
- La dernière permet de le satisfaire si la valeur recherchée est dans le fils droit.

```
> isIn(11,n(12,n(11,f,f),f)).
Réponse : Yes.
> isIn(10,n(12,n(11,f,f),f)).
Réponse : no.
```

Le but `isIn(X,n(12,n(11,f,f),f))` ne produit pas toutes les valeurs présentent dans l'arbre mais seulement la valeur 12. Pourquoi ? Peut-on modifier le programme pour que toutes les valeurs soient produites ? Comment ? Quels est l'inconvénient ?

En supposant que l'arbre est *trié*, réécrivez la spécification du prédicat `isIn` de manière à ce qu'il recherche de manière plus efficace si une valeur est dans un arbre.

Exemple 9

Le prédicat suivant permet l'ajout d'une valeur dans un arbre trié.

```
ajoute(X,f,n(X,f,f)) :-!.
ajoute(X,n(Y,G,D),n(Y,A,D)):- X=<Y,!,ajoute(X,G,A).
ajoute(X,n(Y,G,D),n(Y,G,A)):- ajoute(X,D,A).
```

Comment utiliser ce prédicat pour ajouter la valeur 16 à l'arbre `n(10,n(9,f,f),n(17,f,f))` ? Dessinez le résultat.

Et plus si affinité

Exemple 10

```
chiffre(1). chiffre(2). chiffre(3). chiffre(4). chiffre(5).
chiffre(6). chiffre(7). chiffre(8). chiffre(9). chiffre(0).

puzzle(S,E,N,D,M,O,R,Y) :-
  chiffre(S),chiffre(E),chiffre(N),chiffre(D),
  chiffre(M),chiffre(O),chiffre(R),chiffre(Y),
  S\==0,M\==0,
  S\==E,S\==N,S\==D,S\==M,S\==O,S\==R,S\==Y,
  E\==N,E\==D,E\==M,E\==O,E\==R,E\==Y,
  N\==D,N\==M,N\==O,N\==R,N\==Y,
  D\==M,D\==O,D\==R,D\==Y,
  M\==O,M\==M,M\==Y,
  O\==M,O\==Y,
  R\==Y,
  T1 is D + N*10 + E*100 + S*1000,
  T2 is E + R*10 + O*100 + M*1000,
  T3 is Y + E*10 + N*100 + O*1000 + M*10000,
  T3 is T1 + T2.
```

Que réalise le programme ci-dessus et comment le lance-t-on ?