PROLOG

1.syntaxe :

Un programme Prolog est constitué d'un ensemble de clauses ;

1.a. Les Termes :

Les objets manipulés par un programme Prolog (les "données" du programme) sont appelés des termes:

- les *variables* représentent des objets inconnus de l'univers. Syntaxiquement, une variable est une chaîne alpha-numérique commençant par une majuscule (ex. Var, X, Var_longue_2) ou par un sous-ligné (ex. _objet, _21). Une variable anonyme est notée ``_'' et représente un objet dont on ne souhaite pas connaître la valeur.
- Les *termes constant* (ou termes atomiques) représentent les objets simples connus de l'univers. On distingue trois sortes de termes élémentaires:
    - les nombres: entiers ou flottants,
    - les identificateurs (parfois appelés atomes): un identificateur est une chaîne alpha-numérique commençant par une minuscule (ex. tayeb, aX12, ali),
    - les chaînes de caractères entre guillemets (ex. "Tayeb est \#\{@", "123").

- Les *termes composés* représentent les objets composés (structurés) de l'univers. Syntaxiquement, un terme composé est de la forme: *foncteur(t_1, ..., t_n)* où foncteur est une fonction qui est une chaîne alpha-numérique commençant par une minuscule, et t_1, ..., t_n sont des termes (variables, termes constant ou termes composés). Le nombre d'arguments n est appelé *arité du terme.*

Par exemple, *adresse(18,"rue du 1er novembre ",Ville)* est un terme composé de foncteur *adresse* et d'arité 3, dont les deux premiers arguments sont les termes élémentaires *18 et "rue du 1er novembre"* et le troisième argument est la variable *Ville.*

De même, *cons(a,cons(X,nil))* est un terme composé de foncteur *cons* et d'arité 2, dont le premier argument est le terme élémentaire *a* et le deuxième argument le terme composé *cons(X,nil*).

1.b. Les relations, ou atomes logiques

Un atome logique exprime une relation entre des termes ; cette relation peut être vraie ou fausse. Syntaxiquement, un atome logique est de la forme: *symbole-de-prédicat(t_1, ..., t_n)* où *symbole-de-prédicat* est une chaîne alpha-numérique commençant par une minuscule, et t_1, ..., t_n sont des termes. Le nombre d'arguments n est appelé *arité* de l'atome logique.

Par exemple, *pere(tayeb, ali)* est une relation d'arité 2 entre les termes élémentaires *tayeb* et *ali* pouvant être interprétée par ``tayeb est le père de ali".

De même, *habite(X, adresse(12,"rue amirouche ",alger))*est une relation d'arité 2 entre la variable X et le terme composé *adresse(12,"rue amirouche",alger)* pouvant être interprétée par ``une personne inconnue X habite à l'adresse (*12, "rue amirouche", alger)*".

1.c. Les clauses :

Une clause est une affirmation inconditionnelle (un fait) ou conditionnelle (une règle).

- Un *fait* est de la forme: A. ou A est un atome logique (formule atomique), et signifie que la relation définie par A est vraie (sans condition). Par exemple, le fait *pere(tayeb, ali).* indique que la relation ``*tayeb* est le père de *ali* " est vraie. Une variable dans un fait est quantifiée universellement. Par exemple, le fait *egal(X,X).*indique que la relation ``X est égal à X " est vraie pour toute valeur (tout terme) que X peut prendre.

- Une *règle* est de la forme: A_0 :- A_1 , ... , A_n. où A_0, A_1 , ... , A_n sont des atomes logiques. Une telle règle signifie que la relation A_0 est vraie si les relations A_1 et ... et A_n sont vraies. A_0 est appelé *tête de clause* et A_1, ... , A_n est appelé *corps de clause*.
  Une variable apparaissant dans la tête d'une règle (et éventuellement dans son corps) est quantifiée universellement. Une variable apparaissant dans le corps d'une clause mais pas dans sa tête est quantifiée existentiellement. Par exemple, la clause *meme_pere(X,Y) :- pere(P,X), pere(P,Y).* se lit: ``pour tout X et pour tout Y, me*me_pere(X,Y)* est vrai s'il existe un P tel que *pere(P,X) et pere(P,Y)* soient vrais".

1.d. Les programmes Prolog :

Un programme Prolog est constitué d'une suite de clauses regroupées en paquets. L'ordre dans lequel les paquets sont définis n'est pas significatif. Chaque paquet définit un prédicat et est constitué d'un ensemble de clauses dont l'atome de tête a le même symbole de prédicat et la même arité. L'ordre dans lequel les clauses sont définies est significatif. Intuitivement, deux clauses d'un même paquet sont liées par un « ou » logique. Par exemple, le prédicat personne défini par les deux clauses:

personne(X) :- homme(X).
personne(X) :- femme(X).

se lit ``pour tout X, *personne(X)* est vrai si *homme(X)* est vrai ou *personne(X)* est vrai si *femme(X)* est vrai".

1.e. Exécution de programmes Prolog

``Exécuter" un programme Prolog consiste à poser une question à l'interprète PROLOG. Une question (ou but ou activant) est une suite d'atomes logiques séparés par des virgules. La réponse de Prolog est ``*yes*" si la question est une conséquence logique du programme, ou ``*no*" si la question n'est pas une conséquence logique du programme. Une question peut comporter des variables, quantifiées existentiellement. La réponse de Prolog est alors l'ensemble des valeurs des variables pour lesquelles la question est une conséquence logique du programme.

Par exemple, la question *?- pere(tayeb,X), pere(X,Y).*
se lit ``est-ce qu'il existe un X et un Y tels que *pere(tayeb,X) et pere(X,Y)* soient vrais". La réponse de Prolog est l'ensemble des valeurs de X et Y qui vérifient cette relation. Autrement dit, la réponse de Prolog à cette question devrait être l'ensemble des enfants et petits-enfants de *tayeb*... si *tayeb* est effectivement *grand-père*.

2. Signification (sémantique) d'un programme Prolog

2.a Définitions préliminaires

Rappel Substitution : Une substitution (notée σ) est une fonction de l'ensemble des variables dans l'ensemble des termes. Par exemple, $\sigma = \{ X/Y, Z / f(a,Y) \}$ est la substitution qui ``remplace" *X* par *Y, Z* par *f(a,Y),* et laisse inchangée toute autre variable que *X* et *Z*. Par extension, une substitution peut être appliquée à un atome logique.
Exemple, $\sigma(p(X,f(Y,Z))) = p(\sigma(X),f(\sigma(Y), \sigma(Z))) = p(Y,f(Y,f(a,Y)))$

Rappel Instance : Une instance d'un atome logique A est le résultat $\sigma(A)$ de l'application d'une substitution σ sur *A*. Par exemple, *pere(tayeb,ali)* est une instance de *pere(tayeb,X).*

Rappel Unificateur : Un unificateur de deux atomes logiques *A1* et *A2* est une substitution σ telle que $\sigma(A1) = \sigma(A2)$.

Unificateur plus général : Un unificateur σ de deux atomes logiques *A1* et *A2* est le plus général (upg) si pour tout autre unificateur σ′ de *A1* et *A2*, il existe une autre substitution σ″ telle que $\sigma' = \sigma''(\sigma)$.

Exemple : σ = *{ X / Y }* est un upg de *p(X,b)* et *p(Y,b),* tandis que σ′ = *{ X / a, Y / a }* n'est pas un upg de *p(X,b)* et *p(Y,b).*

Il existe des algorithmes dont l'algorithme de Robinson (unification) qui calcule un upg de deux termes, ou rend "echec" si les deux termes ne sont pas unifiables.

2.b. Dénotation d'un programme Prolog

La dénotation d'un programme Prolog P est l'ensemble des atomes logiques qui sont des conséquences logiques de P. Ainsi, la réponse de Prolog à une question est l'ensemble des instances de cette question qui font partie de la dénotation. Cet ensemble peut être "calculé" par une approche ascendante, dite en chaînage avant: on part des faits (autrement dit des relations qui sont vraies sans condition), et on applique itérativement toutes les règles conditionnelles pour déduire de nouvelles relations ... jusqu'à ce qu'on ait tout déduit.
Considérons par exemple le programme Prolog P suivant:

parent(ali,mohamed).
parent(mohamed,karima).
parent(karima,meriem).
h*omme(ali).*
homme(mohamed).
pere(X,Y) :- parent(X,Y), homme(X).
grand_pere(X,Y) :- pere(X,Z), parent(Z,Y).

L'ensemble des relations vraies sans condition dans P est E_0 :
E_0={parent(ali,mohamed), parent(mohamed,karima), parent(karima,meriem), homme(ali), homme(mohamed) }
à partir de E_0 et P, on déduit l'ensemble des nouvelles relations vraies E_1:
E_1 = { pere(ali,mohamed), pere(mohamed,karima) }
à partir de E_0, E_1 et P, on déduit l'ensemble des nouvelles relations vraies E_2:
E_2 = { grand_pere(ali,karima), grand_pere(mohamed,meriem) }
à partir de E_0, E_1, E_2 et P , on ne peut plus rien déduire de nouveau. L'union de E_0 , E_1 et E_2 constitue la dénotation (l'ensemble des conséquences logiques) de P.

Malheureusement, la dénotation d'un programme est souvent un ensemble infini et n'est donc pas calculable de façon finie. Considérons par exemple le programme P suivant:

plus(0,X,X).
plus(succ(X),Y,succ(Z)) :- plus(X,Y,Z).

L'ensemble des atomes logiques vrais sans condition dans P est
E_0 = { plus(0,X,X) }
à partir de E_0 et P, on déduit
E_1 = { plus(succ(0),X,succ(X)) }

à partir de E_0, E_1 et P, on déduit
E_2 = { plus(succ(succ(0)),X,succ(succ(X))) }
à partir de E_0, E_1, E_2 et P , on déduit
E_3 = { plus(succ(succ(succ(0))),X,succ(succ(succ(X)))) }
etc ...

2.c. Signification opérationnelle

D'une façon générale, on ne peut pas calculer l'ensemble des conséquences logiques d'un programme par l'approche ascendante: ce calcul serait trop couteux, voire infini. En revanche, on peut démontrer qu'un but (composé d'une suite d'atomes logiques) est une conséquence logique du programme, en utilisant une approche descendante, dite en chaînage arrière:

Pour prouver un but composé d'une suite d'atomes logiques (*But = [A_1, A_2, .., A_n]),* l'interprête Prolog commence par prouver le premier de ces atomes logiques (*A_1*). Pour cela, il cherche une clause dans le programme dont l'atome de tête s'unifie avec le premier atome logique à prouver (par exemple, la clause *A'_0 :- A'_1, A'_2, ..,A'_r* telle que *upg(A_1,A'_0)* = σ) puis l'interprête Prolog remplace le premier atome logique à prouver (*A_1*) dans le but par les atomes logiques du corps de la clause, en leur appliquant la substitution σ. La nouveau but à prouver devient But = [σ(*A'_1),* σ*(A'_2), ..,* σ*(A'_r),* σ*(A_2), .., *σ*(A_n)]*

L'interprête Prolog recommence alors ce processus, jusqu'à ce que le but à prouver soit vide, c'est à dire jusqu'à ce qu'il n'y ait plus rien à prouver. A ce moment, l'interprête Prolog a prouvé le but initial ; si le but initial comportait des variables, il affiche la valeur de ces variables obtenue en leur appliquant les substitutions successivement utilisées pour la preuve
Il existe généralement plusieurs clauses dans le programme Prolog dont l'atome de tête s'unifie avec le premier atome logique à prouver. Ainsi, l'interprête Prolog va successivement répéter ce processus de preuve pour chacune des clauses candidates. Par conséquent, l'interprête Prolog peut trouver plusieurs réponses à un but.

Ce processus de preuve en chaînage arrière est résumé par la fonction *prouver(But)* suivante. Cette fonction affiche l'ensemble des instances de But qui font partie de la dénotation du programme:

Procedure *prouver(But: liste d'atomes logiques , but-init)*
si *But = []*
alors */* le but initial est prouvé, afficher les valeurs des variables du but initial */*
Sinon *soit But = [A_1, A_2, .., A_n]*
pour *toute clause (A'_0 :- A'_1, A'_2, ..,A'_r) du programme (les var ont été renommées)*
calculer σ *qui est l' upg(A_1 ,A'_0)*
si σ ≠ *echec*
alors *prouver([*σ *(A'_1),* σ *(A'_2), ..* σ *(A'_r),* σ*(A_2), .. *σ*(A_n)],* σ *(But-init))*
finsi
finpour
finsi
fin prouver

Quand on pose une question à l'interprète Prolog, celui-ci exécute dynamiquement l'algorithme précédent. L'arbre constitué de l'ensemble des appels récursifs à la procédure prouver_bis est appelé *arbre de recherche*.

Remarques

- la stratégie de recherche n'est pas complète, dans la mesure où l'on peut avoir une suite infinie d'appels récursifs,
- la stratégie de recherche dépend d'une part de l'ordre de définition des clauses dans un paquet (si plusieurs clauses peuvent être utilisées pour prouver un atome logique, on considère les clauses selon leur ordre d'apparition dans le paquet), et d'autre part de l'ordre des atomes logiques dans le corps d'une clause (on prouve les atomes logiques selon leur ordre d'apparition dans la clause).

3. Les listes :

La liste est un terme composé particulier de symbole de fonction ``." et d'arité 2: le premier argument est l'élément de tête de la liste, et le deuxième argument est la queue de la liste. La liste vide est notée ``[]".

Notations :

- la liste .(X,L) est également notée [X|L] ,
- la liste .(X1, .(X2, L)) est également notée [X1, X2|L],
- la liste .(X1, .(X2, ..., .(Xn, L) ...)) est également notée [X1, X2, ..., Xn|L],
- la liste [X1, X2, X3, ..., Xn|[]] est également notée [X1, X2, X3, ..., Xn].

Par exemple, la liste [a,b,c] correspond à la liste .(a,.(b,.(c,[]))) et contient les 3 éléments a , b et c. La liste [a,b|L] correspond à la liste .(a,.(b,L)) et contient les 2 éléments a et b , suivis de la liste (inconnue) L.

Une liste est une structure récursive: la liste Liste = [X|L] est composée d'un élément de tête X et d'une queue de liste L qui est elle-même une liste. Par conséquent, les relations Prolog qui ``manipulent" les listes seront généralement définies par

- une ou plusieurs clauses récursives, définissant la relation sur la liste [X|L] en fonction de la relation sur la queue de liste L,
- une ou plusieurs clauses non récursives assurant la terminaison de la manipulation, et définissant la relation pour une liste particulière (par exemple, la liste vide, ou la liste dont l'élément de tête vérifie une certaine condition...).

4. La coupure

4.a. Signification opérationnelle de la coupure

La coupure, aussi appelée "cut", est notée !. La coupure est un prédicat sans signification logique (la coupure n'est ni vraie ni fausse), utilisée pour "couper" des branches de l'arbre de recherche. La coupure est toujours "prouvée" avec succès dans la procédure *prouver* décrite au paragraphe 2.c. La "preuve" de la coupure a pour effet de bord de modifier l'arbre de recherche: elle coupe l'ensemble des branches en attente créées depuis l'appel de la clause qui a introduit la coupure.

Considérons par exemple le programme Prolog suivant:

```
p(X,Y) :- q(X), r(X,Y).
p(c,c1).
q(a).
q(b).
r(a,a1).
r(a,a2).
```

r(b,b1).
r(b,b2).
r(b,b3).

L'arbre de recherche construit par Prolog pour le but p(Z,T) est:

prouver([p(Z,T)])
b1 : σ = {Z / X, T / Y}
---- prouver([q(X),r(X,Y)])
b11 : σ = {X / a}
----- prouver([r(a,Y)])
b111 : σ = {Y / a1}
------ prouver([],[p(a,a1)] --> solution1 = {Z=a, T=a1}
b112 : σ = {Y / a2}
------ prouver([]) --> solution2 = {Z=a, T=a2}
b12 : σ = {X / b}
------ prouver([r(b,Y)])
b121 : σ = {Y / b1}
------ prouver([]) --> solution3 = {Z=b, T=b1}
b122 : σ = {Y / b2}
------ prouver([]) --> solution4 = {Z=b, T=b2}
b123 : σ = {Y / b3}
------ prouver([]) --> solution5 = {Z=b, T=b3}
b2 : σ = {Z / c, T / c1}
---- prouver([]) --> solution6 = {Z=c, T=c1}

En fonction de l'endroit où l'on place une coupure dans la définition de p, cet arbre de recherche est plus ou moins élagué, et certaines solutions supprimées :

Si on définit p par *p(X,Y) :- q(X), r(X,Y), !.*
p(c,c1).
Prolog donne la solution 1 puis coupe toutes les branches en attente (b112, b12 et b2).

si on définit p par *p(X,Y) :- q(X), !, r(X,Y).*
p(c,c1).
Prolog donne les solutions 1 et 2 puis coupe les branches en attente (b12 et b2).

si on définit p par : *p(X,Y) :- !, q(X), r(X,Y).*
p(c,c1).
Prolog donne les solutions 1, 2, 3, 4 et 5 puis coupe la branche en attente (b2).

4.b. Les applications de la coupure

- Recherche de la première solution
- Exprimer le déterminisme
- La négation
- Le "si-alors-sinon"