Prolog - EISTI - ING 2

Yannick Le Nir

Ecole Internationale des Sciences du Traitement de l'Information

yannick.lenir@eisti.fr

Sommaire

Avant Propos

Documents annexes

L'ensemble des documents liés au cours de Prolog se trouve sur le site `http ://vfedu.eisti.fr`.
Vous y trouverez le polycopié de Chrysostome Baskiotis et Maria Malek dont est issue cette présentation, ainsi qu'une bibliographie, le planning et le programme détaillé de ce cours.

Paradigmes de programmation

Classification

- Langages procéduraux ou impératifs
  Fortran, Pascal, C, Ada
- Langages fonctionnels
  Lisp, Simula, ML, Scheme, Caml
- Langages relationnels ou déclaratifs
  Prolog

Langages fondés sur la Logique propositionnelle

Caractéristiques

- Faits du domaine de connaissance = données
- Règles pour exprimer les relations entre faits
- Déductions pour répondre aux questions concernant le domaine de connaissance

Comparaison des langages

Formules

- N. Wirth
  Algorithmes + Structure de données = Programmes
- R. Kowalski
  Algorithme = Logique + Contrôle

Prolog

- Logique : Programmeur
- Contrôle : Prolog

Utilisation

Industrie

- En dents de scie
  - Echec du projet Japonais d'ordinateurs de 5eme génération
  - Succès de la récente BDD du génôme humain
- Peu connu (tentative en 1980 avec le Logo par éducation nationale)

Sommaire

Introduction

Logique des propositions

Généralités

Les Faits

Les règles

Les données

Exécution

Les éléments de base de Prolog

Logique des prédicats, Liste et Récursivité

Prédicats de Prolog

Listes et récursivité

Techniques de programmation

Base de données

Requêtes

Programmation logique

Gestion de la base de données

Bases de données

Modélisation des connaissances

Univers du discours

- Environnement donné : $\mathcal{U}$
- Triplet : (objet,attribut,valeur)
  - objet : entités physiques ou conceptuelles de l'univers du discours
  - attributs : caractéristiques/propriétés des objets et relations
  - valeur : spécification d'un attribut pour un objet particulier

Programmation Prolog

Etapes

- Spécification des faits concernant les objets
- Construction des règles concernant les relations entre objets
- Réponse à des questions posées concernant l'existance des objets et/ou relations entre objets

Syntaxe

- Faits, règles : expressions logiques
- Constantes : termes logiques commençant par une minuscule
- Variables : termes logiques commençant par une majuscule

Faits

Utilité

- Existence d'un objet particulier
  exemple : `cube(toto,bleue).`
- Valeur numérique ou symbolique
  exemple : `volumeCube(toto,100).`

Forme générale : `nomFait(objet1,objet2,...,objetn).`
(le point "." indique la fin du fait)

Faits (suite)

Propriétés

- Noms de Faits identiques + nb arguments différents = Relations différentes
- Conjonction de Faits ("et") : Fait1,Fait2
- Disjonction de Faits ("ou") : Fait1 ;Fait2
- Analogie Faits/Données des langages de 3ieme génération
- nomFait est toujours un prédicat (vrai si les arguments se trouvent dans la base de données)

Règles

Description

- Manière dont un fait est relié à d'autres faits (axiomes ou théorèmes logiques de la logique computationnelle)
- Equivalent aux clauses de Horn (comme les Faits)

Syntaxe

Forme générale : `A:- B1,B2,...,Bn`
où `A` (entête = atome positif), `Bk` (corps = littéraux) sont des faits.

Base de données

Définition

Les faits et règles d'un univers du discours constituent la base de données de cet univers (base de connaissances de l'environnement)

Propriétés

- Analogie Règles/Sous-programmes ou fonctions des langages de 3ieme génération

Questions (But = clause avec entête vide)

Rôle

Parcours de la base de données pour vérifier si la question est filtrée

- soit par les faits définis dans la base de données
- soit par un fait issu par application des règles sur les faits de la base de données

Réponse à la question liée à la possibilité qu'elle soit une conséquence logique de la base de données

Syntaxe

```
nomRelation(objet1,objet2,...,objetn)?
```

Données

Différents types

Données = Termes de la logique computationnelle

- Données simples (non typées) :
  - constantes (commencent par une minuscule ou entre ' ') :
    - nombres (entiers ou réels)
    - atomes (toto, 'Toto')
  - variables (commencent par une majuscule ou par _) :
    - variable anonyme (_)
- Données structurées depuis la base de données (non typées)
  - Foncteurs :
    `adresse('Toto',26,av,'des lilas',64000,'Pau')`
  - Prédicats :
    `couleurRouge(titre(TitreLivre),couleur(Couleur))`
    ici le prédicats est vrai si Couleur est unifié avec rouge

Exécution

Langage interprété

- L'ordinateur exécute immédiatement les commandes saisies :
  traduction immédiate en langage machine
- Moteur d'inférence : teste si les expressions sont vraies ou fausses (unification par backtracking sur la base de faits)

Exécution (suite)

Exécution

1. Lancer l'interpréteur Prolog
2. Charger le programme :
   `?-consult ('nomProgramme.pl').`
3. Poser la question
4. Prolog fournit une réponse par application de l'algorithme de résolution SLD (sans vérification des occurrences)
   - Si Prolog arrive à filtrer la question, il affiche `yes` et le contenu des variables instanciées.
   - On peut obtenir les autres réponses possibles en tapant ; (ou)
   - On peut arréter en tapant sur la touche "Entrée"

Opérateurs

Egalité et comparaison

- `=:` unifie deux termes
- `==:` vérifie si deux termes sont identiques
- `<:` vérifie si un terme est plus petit qu'un autre
- `=<:` vérifie si un terme est plus petit ou égal à un autre
- `>:` vérifie si un terme est plus grand qu'un autre
- `>=:` vérifie si un terme est plus grand ou égal à un autre
- `\=:` teste la non-unification de deux termes
- `\==:` teste si deux termes ne sont pas identiques

Opérateurs (suite)

Arithmétiques

- `=:=` : égal
- `<` : plus petit
- `=<` : plus petit ou égal
- `>` : plus grand
- `>=` : plus grand ou égal
- `=\=` : différent
- `is` : évaluation d'une expression et assignation à une variable

Sommaire

Prédicats de base

Entrées-sorties

- `print(X), print('toto')` : afficher le contenu de X ou le mot "toto"
- `tab(5)` : affiche 5 espaces blancs
- `nl` : saut d'une ligne
- `read(X)` : lecture d'une valeur et stockage dans X

Définir des opérateurs

```
:- op(priorité, type, nom)
```

- priorité : valeur comprise entre 1 (+ prioritaire) et 32000
- type : infixe (xfx,xfy,yfx,yfy), préfixe (fx,fy), postfixe (xf,yf)
- nom : nom de l'opérateur

Prédicats extralogiques

Première catégorie

- `var(X)` : X est une variable
- `atom(X)` : X est un nom d'une constante
- `number(X)` : X est un nombre

Seconde catégorie

- `assert(X)` : Ajouter le fait X à la base
- `asserta(X)` : Ajouter le fait X au début
- `assertz(X)` : Ajouter le fait X à la fin
- `retract(X)` : Supprime toutes les occurrences de X

Représentation des listes

Définition

Suite de termes séparés par des virgules. Possibilité de contenir des sous listes.

Exemple :

```
L=[toto,[26 ,[av_des_lilas],[pau]],vive_prolog]
```

Utilisation

Accès à la tête et au reste de la liste : `L=[Tete|Reste]`

Exemple : `Test=toto` et

`Reste=[[26 ,[av_des_lilas],[pau]],vive_prolog]`

Parcours

De manière séquentielle sur tous les éléments de la liste, en ajoutant un élément vide `[]` comme dernier élément pour s'arréter.

Principe de Récursivité

Cas d'arrêts

- Liste vide
- Valeurs numériques atteintes
- ...

Exemple : `longueur([],0).`

Cas récursifs

Appel récursif du prédicat lui même

Exemple :

```
longueur([X|Y],N):- longueur(Y,N1),N is N1+1.
```

Récursivité sur fonctions numériques

Programme somme

```
somme(0,0).
somme(N,Somme):- N>0,N1 is N-1,
                 somme(N1,Somme1),Somme is Somme1+N1.
```

- Arrêt : comparaison de la valeur d'une variable avec une valeur fixée qui servira d'initialisation (ici N et 0)
- Récursivité : Appel avec un argument plus simple, plus proche de la valeur d'initialisation du cas d'arrêt (ici $N1$)

Récursivité simple

Programme longueur : arrêt liste vide

```
longueur([],0).
longueur([X|Y],N):- longueur(Y,N1), N is N1+1.
```

Programme affiche nième : arrêt position spécifique atteinte

```
affn(1,[X|_]):- write(X),nl.
affn(N,[Tete|Reste]):- N>1,N1 is N-1,affn(N1,Reste).
```

Récursivité simple(suite)

Programme membre : arrêt élément trouvé

```
membre(X, [X | _]).
membre(X, [Tete | Reste]) :- membre(X,Reste).
```

On peut gérer un élément non présent dans la liste, en utilisant l'opérateur "ou".
Exemple :

```
toto :- (membre(a,[b,a,c]),write('succes'));
            (write('echec')),nl.
```

Récursivité multiple

Programme recherche : arrêts multiples

```
membreT(X,[X|_]).
membreT(X,[Tete|Reste]):- atom(Tete),membreT(X,Reste).
membreT(X,[Tete|Reste]):- not(atom(Tete)),
                          (membreT(X,Tete);membreT(X,Reste)).
```

Programme tri : plusieurs listes

```
tri([X|Xs ),Y) :- partition(Xs,X,P,G),tri(P,Ps),
                  tri(G,Gs),conc(Ps,[X|Gs],Y).
tri([],[]).
partition([X|Xs],Y,[X|P],G):- X<Y,partition(Xs,Y,P,G).
partition([X|Xs],Y,P,[X|G]):- X>=Y,partition(Xs,Y,P,G).
partition([],Y,[],[]).
```

Coupure

Fonctionnement

Symbole cut (!), permettant de couper l'évaluation de l'arbre SLD dès qu'une clause est vérifiée

Exemple

```
B :- C1, C2, ..., Ci, ! , C(i+1), ... , Cn.
B :- D1, D2, ..., Dm.
```

- Si une des conditions `C1,..,Ci` n'est pas satisfaite, alors Prolog passe à la clause suivante.
- Si toutes les conditions `C1,...,Ci` sont satisfaites, alors le cut est atteint, et en tant que but, est réalisé.
- Seul le backtracking sur les clauses `C(i+1),...,Cn` est alors permis.

Exemple

Programme

```
(1) p(a). (2) p(X) :- q(X),r(X). (3) p(X) :- u(X).
(4) q(a). (5) q(b).              (6) q(c).
(7) r(a). (8) r(b).              (9) r(d).
(10) u(d).
```

- `?-p(X)` donne `X = a, X = a, X = b, X = d`
- Si on rempace (1) par `p(a):-!`, la seule réponse est `X = a`
- Si on remplace (2) par `p(X) :- !,q(X),r(X).`, les réponses sont `X = a, X = a, X = b`
- Sin on remplace (2) par `p(X) :- q(X),!,r(X).`, les réponses sont `X = a, X = a`

Coupure

Règles à suivre

- Ajout de cut pour éviter de parcourir des branches de calculs inutils ou superflus.
- Placer le cut le plus proche possible du début de clause
- Ne pas utiliser le cut à la dernière clause d'une primitive

Deux types de cut

- cut vert : élagage des branches sue lesquelles il n'y a pas de solution sans modifier la sémantique du programme
- cut rouge : élagage de branche sur lesquelles il peut y avoir des solutions, pouvant donc modifier la sémantique du programme

Exemple

Programme minimum

```
minimum(X,Y,X) :- X =< Y.
minimum(X,Y,Y) :- X > Y.
```

Programme minimum cut vert

```
minimum(X,Y,X) :- X =< Y, !.
minimum(X,Y,Y) :- X > Y.
```

Programme minimum cut rouge

```
minimum(X,Y,X) :- X =< Y, !.
minimum(X,Y,Y).
```

Cut rouge

Exemple variables dégénérées

```
(1) intersection([],_,[]).
(2) intersection([H|T],L,[H|T2])
                :- member(H,L), intersection(T,L,T2).
(3) intersection([_H|T],L,T2):- intersection(T,L,T2).
```

- `intersection([a,b,c],[a,b,d],R).` donne `R=[a,b],R=[a], R=[b], R=[].`
- Si on remplace (2) par

  ```
  intersection([H|T],L,[H|T2])
                  :- member(H,L),!,intersection(T,L,T2).
  ```

  on obtient bien `X=[a,b].`

Négation en utilisant le cut

Prédicat différent

```
dif(X,X) :- !, fail.
dif(X,Y).
```

Négation non

```
non(But) :- call(But),!,fail.
non(But).
```

- Le prédicat `call(X)` permet de transformer un terme X en un but pour prolog.
- Attention, à utiliser uniquement avec des constantes ou des variables déjà instanciées.

Inconvénients de la négation

Attention

Soit la base de données suivantes :

```
innocent(toto).
occupation(jojo,ailleurs).
coupable(koko).
innocent(X) :- occupation(X,ailleurs).
coupable(X) :- occupation(X,ici).
```

- La question `innocent(fourier).` répond `non`.
- Pour améliorer, on peut rajouter `coupable(X):-non(innocent(X)).`
- La question`coupable(fourier).` répond alors `oui`.

Autre exemple problématique

Attention

Soit la base de données suivante :

```
voiture(rouge).
voiture(verte).
decapotable(rouge).
berline(X) :- non(decapotable(X)).
```

- La question `voiture(X),berline(X).` répond `oui`
- alors que la question `berline(X),voiture(X).` répond `non`.
- Le problème vient du fait que dans le second cas, la variable `X` n'est pas instanciée lors de l'appel du prédicat `berline(X)`.
- Il faut donc n'utiliser le `non` que sur des constantes ou des variables déjà instanciées.

Base de données relationnelles

Relations

- exprimée par une table :
  - colonne = attribut de la relation
  - ligne = enregistrement
- prolog :
  - table = prédicat
  - nombre colonne = arité du prédicat
  - colonne i = argument i
  - enregistrement = fait

Exemples

Relation pere-fils

Deux enregistrements de la table père :

```
pere(toto,titi).
pere(toto,koko).
```

Prédicat père d'arité 2

Relation mere

```
mere(lola,titi).
mere(lola,koko).
```

Requêtes

Requêtes simples

Prédicat constituant la base de faits :

```
?pere(toto,X). /* donne tous les enfants de toto. */
?pere(X,koko). /* donne le père de koko.          */
```

Requêtes composées

Ensemble de requêtes simples reliées par conjonction et disjonction :
Règle dont le corps est la conjonction des requêtes à poser.
et la tête est la définition simplifiée de la requête.

Requête composée

Requête pere et mere

```
parent(toto,titi).
parent(toto,koko).
parent(lola,titi).
parent(lola,koko).
homme(toto).
homme(koko).
homme(titi).
femme(lola).
pere(X,Y):-parent(X,Y),homme(X).
mere(X,Y):-parent(X,Y),femme(X).
```

Requêtes (conjonction de requêtes simples) :

```
?pere(X,koko). /* trouver le parent de koko X et X est un homme  */
?mere(X,koko). /* trouver le parent de koko X et X est une femme */
```

Requête composée

Requête parent

```
pere(toto,titi).
pere(toto,koko).
mere(lola,titi).
mere(lola,koko).
parent(X,Y):-pere(X,Y).
parent(X,Y):-mere(X,Y).
```

Requêtes (disjonction de requêtes simples) :

```
?parent(X,koko). /* trouver le père ou la mère de koko */
```

Requêtes récursives

Cas particulier de requête composée

Relation qui s'appelle dans le corps d'une clause :

```
ancetre(X,Y):-parent(X,Z),parent(Z,Y).
ancetre(X,Y):-parent(X,Z),ancetre(Z,Y).
```

Algèbre relationnelle

Opérations

- l'union
- la différence ensembliste
- le produit cartésien
- la projection
- la sélection

Opération union

Définition

Création d'une relation d'arité n à partir de deux relations r et s d'artité n.

```
r_union_s(X1,...,Xn) :- r(X1,...,Xn).
r_union_s(X1,...,Xn) :- s(X1,...,Xn).
```

Exemple : la relation parent définie précédemment

Opération de différence

Définition

La différence ensembliste est exprimée à l'aide de la négation

```
r_diff_s(X1,...,Xn) :- r(X1,...,Xn), not s(X1,...,Xn).
r_diff_s(X1,...,Xn) :- s(X1,...,Xn), not r(X1,...,Xn).
```

Exemple : la relation parent définie précédemment

Opération du produit cartésien

Définition

Opération permettant d'obtenir une relation à partir d'une relation d'arité n et d'une relation d'arité m

```
r_prod_s(X1,...,Xm,...,Xm+n)
        :- r(X1,...,Xm), s(Xm+1,...,Xm+n).
```

Opération de projection

Définition

Opération permettant d'obtenir une relation contenant un sous-ensemble des attributs composant la relation de départ.
Exemple relation $r13$: projection de r selon le premier et le troisième argument.

```
r13(X1,X3) :- r(X1,X2,X3).
```

Opération de sélection

Définition

Opération permettant d'obtenir une relation dérivée à partir d'une relation en posant certaines contraintes sur certaines données

```
r1(X1,X3) :- r(X1,X2,X3), X2 > X3.

r2(X1,X3) :- r(X1,X2,X3), r3(X1).
r3(const1).
r3'const2).
```

Structure de la base

Relations

- $r_1, \cdots, r_n$: ensemble des relations
- cle_i : attribut clé pour chaque relation.
- att_i^j : attribut numéro j de la relation r_i

BDD

Prédicat bddVirt d'arité 5 :

1. nom de la relation
2. nom de la clé
3. valeur de la clé
4. nom de l'attribut j
5. valeur de l'attribut j

Base de données déductive

Principe

- Utilisation des règles pour enrichir les bases de données avec de nouvelles données.
- Exemple : ajout du schéma relationnel ancetreR avec les données associées
- Les faits ancetreR sont des faits déduits :

```
q2(X,Y):-ancetre(X,Y),assert(ancetreR(X,Y)).
```

- Tous ces faits sont démontrables à partir du programme principal, par le modus-ponens etl'instanciation des variables.
- L'ensemble des faits obtenu correspond au modèle minimal de Herbrand (ensemble de départ = faits filtrés)