Introduction à la programmation logique

Cours « Découverte de l'intelligence artificielle »

Nicolas Delestre

INSA
ROUEN NORMANDIE

Plan

1. Recherche exploratoire des solutions : du comment à quoi
2. La programmation logique
   - Un nouveau paradigme de programmation
   - Prolog
3. Vocabulaire
4. Les termes complexes
   - Les opérateurs
   - Les expressions arithmétiques
5. Les listes
6. Conclusion

INSA ROUEN NORMANDIE

Rappel : les méthodes exploratoires classiques

Approches étudiées précédemment

- **Force brute** : explorer exhaustivement toutes les solutions possibles
- **Minmax** (pour les jeux à sommes nulles) : exploration de toutes les solutions impossibles, se limiter à l'exploration de quelques coups et partir du principe que l'adversaire joue au mieux (**alpha-beta** : élagage de l'arbre)
- **Monte-Carlo** : exploration de toutes les solutions impossibles, donc explorer aléatoirement mais statistiquement l'espace de recherche pour approcher une "bonne" solution

Constat

Ces méthodes imposent de programmer explicitement le **comment explorer** l'espace des solutions.

Limites des approches procédurales

Difficultés rencontrées

- Implémentation compliquée, souvent longue et laborieuse
- Difficulté importante à débugger : identification difficile des erreurs dans l'exploration
- Adaptation limitée à d'autres problèmes, nécessitant souvent des réécritures complètes ou importantes

Illustrations concrètes

- Modifier une IA Minmax de Puissance 4 pour jouer aux échecs exige de nombreux ajustements.
- Adapter l'approche Monte-Carlo du jeu 2048 à d'autres jeux complexes comme le Go nécessite un retravail approfondi.

INSA ROUEN NORMANDIE

Changer de perspective : du **comment** au **quoi**

Idée clé de la programmation logique

- Plutôt que de coder explicitement comment explorer l'espace de recherche, il est préférable d'exprimer ce que constitue une solution
- La recherche elle-même sera confiée à un **moteur logique**

Changement de paradigme

On passe ainsi d'un paradigme **impératif** (comment) à un paradigme **déclaratif** (quoi)

INSA ROUEN NORMANDIE

Paradigmes de programmation 1 / 2

Rappels

- Un paradigme de programmation est la façon d'aborder la résolution d'un problème
- Un paradigme de programmation repose sur des principes, des méthodes et outils

Paradigme de programmation impérative

- Principe : problème et solution sont représentés par des états (état initial et état final) et un programme représente le **comment** passer de l'état initial à l'état final
- Outil : l'affectation permet de passer d'un état i à un état $i + 1$
- Méthode : les schémas (séquentiel, conditionnel et itératif) permettent d'organiser les affectations

INSA ROUEN NORMANDIE

Paradigmes de programmation 2 / 2

Paradigme de programmation structurée

- Sous paradigme de la programmation impérative, il ajoute deux nouveaux outils et une nouvelle méthode
- Outil : sous-programmes (fonction et procédure)
- Méthode : passage de paramètre (association entre paramètres effectifs et paramètres formels)

INSA ROUEN NORMANDIE

Paradigme de la programmation logique

Paradigme de programmation déclarative

- Principe : on décrit le problème, le **quoi** et pas le comment
- Outil : un moteur interprète le problème pour trouver/construire la solution
- Méthode : *dépendant du sous paradigme*

Paradigme de programmation logique

- Sous paradigme de la programmation déclarative
- Principe : on décrit le problème en terme de
  - faits : formules logiques avec uniquement des atomes, (identifiants des éléments du monde à représenter) ou des constantes)
  - règles : formules logiques faisant intervenir des variables
  - une question : formule logique
- Outil : trouver pour quels atomes et/ou quelles constantes, l'interprétation de la question est vraie
- Méthode : algorithmes de démonstration

Prolog 1 / 2

- Langage du paradigme de la PROgrammation LOGique
- Langage inventé par Alain Colmerauer et Philippe Roussel vers 1972 (Marseille)
- « [...]le but n'était pas de faire un langage de programmation mais de traiter les langages naturels, en l'occurrence le Français. »

Fondement théorique

- Fondé sur les clauses de Horn de la logique des prédicats du premier ordre
- Les concepts fondamentaux sont le chaînage arrière, l'unification, la récursivité et le backtracking

INSA ROUEN NORMANDIE

Prolog 2 / 2

Principe

- Prolog permet au programmeur de déclarer des faits, des règles et de répondre à des questions

Éléments de base du langage

- Les clauses de Horn :
  - $r_1 \wedge r_2 \wedge \cdots \wedge r_n \rightarrow h$ avec $n \in \mathbb{N}$
  - En prolog, elles peuvent représenter aussi bien des faits que des règles
- Les atomes (commencent par une minuscule, ou entre simples côtes si utilisation de l'espace ou ne commençant pas par une minuscule) et les variables (commencent par une majuscule)
- Une question

- Il existe plusieurs syntaxes (ISO-Prolog, Edinburgh prolog, etc.)
- Tous les exemples de ce cours utilisent *swi-prolog*

Introduction de la syntaxe 1 / 3

- Fait :

```
fait(atome_ou_constante1,atome_ou_constante2,...,atome_ou_constante3).
```

- Règle :

```
regle(Var_ou_atome_ou_cons01, Var_ou_atome_ou_cons02, ...):-
                cond1(Var_ou_atome_ou_cons11, Var_ou_atome_ou_cons12, ...),
                cond2(Var_ou_atome_ou_cons21,Var_ou_atome_ou_cons22,..),
                ...,
                condn(Var_ou_atome_ou_consn1,Var_ou_atome_ou_consn2,...).
```

Remarques

- Un prédicat est un ensemble de clauses (faits et/ou règles) ayant le même nom et la même arité
- Deux prédicats de même nom mais d'arités différentes sont différents

Introduction de la syntaxe 2 / 3

Exemple : des faits

```
1 /* homme(X) est vrai si X est un homme */
2 homme(patrick).
3 homme(gerard).
4 homme(louis).
5 homme(pierre).
6
7 /* femme(X) est vrai si X est une femme */
8 femme(therese).
9 femme(sandrine).
10 femme(muriel).
11 femme(germaine).
12 femme(yvette).
13
14 /* enfant_parents(Enfant,Parent1,Parent2) est vrai si Enfant est un enfant de Parent1 et Parent2 */
15 enfant_parents(gerard,germaine,louis).
16 enfant_parents(therese,yvette,pierre).
17 enfant_parents(patrick,gerard,therese).
18 enfant_parents(muriel,gerard,therese).
19 enfant_parents(sandrine,gerard,therese).
20 enfant_parents(astride,jean,therese).
```

INSA ROUEN NORMANDIE

Introduction de la syntaxe 3 / 3

Exemple : des règles

```
22 /* enfant_parent(Enfant,Parent) est vrai si Enfant est un enfant de Parent */
23 enfant_parent(Enfant,Parent) :- enfant_parents(Enfant,Parent,_).
24 enfant_parent(Enfant,Parent) :- enfant_parents(Enfant,_,Parent).
25
26 /* pere_enfant(Pere,Enfant) est vrai si Pere est pere de Enfant */
27 pere_enfant(Pere,Enfant) :- homme(Pere), enfant_parent(Enfant,Pere).
28
29 /* mere_enfant(Mere,Enfant) est vrai si Mere est mere de Enfant */
30 mere_enfant(Mere,Enfant) :- femme(Mere), enfant_parent(Enfant,Mere).
31
32 /* grandpere_petitenfant(GrandPere,Enfant) vrai si GrandPere est grand-pere de Enfant */
33 grandpere_petitenfant(GrandPere,Enfant) :- pere_enfant(GrandPere,Parent), pere_enfant(Parent,Enfant).
34 grandpere_petitenfant(GrandPere,Enfant) :- pere_enfant(GrandPere,Parent), mere_enfant(Parent,Enfant).
35
36 /* grandmere_petitenfant(GrandPere,Enfant) vrai si GrandPere est grand-mere de Enfant */
37 grandmere_petitenfant(GrandPere,Enfant) :- mere_enfant(GrandPere,Parent), pere_enfant(Parent,Enfant).
38 grandmere_petitenfant(GrandPere,Enfant) :- mere_enfant(GrandPere,Parent), mere_enfant(Parent,Enfant).
```

INSA ROUEN NORMANDIE

L'interpréteur de swi-prolog 1 / 3

- On lance l'interpréteur à l'aide de la commande *swipl* (ou *pl*)
- On pose alors des « questions » en utilisant un prédicat ou plusieurs prédicats (séparés par des virgules) suivis d'un point
- Un programme prolog (faits et règles) est stocké dans un fichier dont le nom commence par une minuscule et a l'extension `.pl`
- On charge un programme en mettant le nom du fichier entre accolade suivi d'un point
- Quelques prédicats particuliers :
  - *halt/0* pour quitter
  - *help/1* pour obtenir de l'aide

INSA ROUEN NORMANDIE

L'interpréteur de swi-prolog 2 / 3

- Lorsque plusieurs solutions existent, l'interpréteur en affiche une et demande ce qu'il doit faire pour le reste
  - ? pour plus d'information
  - ; ou espace pour la solution suivante
  - a ou entrée pour arrêter l'affichage des solutions

```
$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [genealogie].
true.

?- pere_enfant(gerard,patrick).
true .

?- pere_enfant(therese,patrick).
false.
```

L'interpréteur de swi-prolog 3 / 3

```
?- pere_enfant(X,patrick).
X = gerard ;
false.

?- pere_enfant(gerard,X).
X = patrick ;
X = muriel ;
X = sandrine ;
false.

?- grandpere_petitenfant(louis,patrick).
true .

?- grandpere_petitenfant(germaine,patrick).
false.

?- grandpere_petitenfant(X,patrick).
X = louis ;
X = pierre ;
false.

9 ?- grandpere_petitenfant(louis,X).
X = patrick ;
X = muriel ;
X = sandrine ;
false.
```

Précisions concernant la syntaxe et le vocabulaire 1 / 4

Termes

- Éléments de base du langage
- Deux types de termes : les termes simples et termes complexes

Terme simple : atome, constante ou variable

- Atome :
  - séquence de lettres (majuscule ou minuscule), de chiffres ou du tiret bas, commençant par une minuscule
  - séquence de caractères entourée de guillemets simples
  - séquence de caractères spéciaux (`+`, `-`, `*`, `/`, `<`, `>`, `=`, `:`, `.`, `@`, `~`)
- Constante :
  - chaine de caractères entourée de guillemets doubles
  - nombre : entier ou flottant par défaut représenté en décimale. Pour les entiers, possibilité d'utiliser le `_` comme séparateur : `1_000_000`

Précisions concernant la syntaxe et le vocabulaire 2 / 4

Terme simple : atome, constante ou variable (suite)

- Variable : séquence de lettres (majuscule ou minuscule), de chiffres ou du tiret bas, commençant par une majuscule ou un tiret bas (le tiret bas seul est appelé variable anonyme)

Terme complexe

- Association d'un foncteur (un atome) et d'arguments (n'importe quel type de terme) entre parenthèses, séparés des virgules
- L'arité d'un terme complexe est son nombre d'arguments

Atome logique

- Terme complexe particulier
- Met en relation logique les termes
- Composant des clauses

Précisions concernant la syntaxe et le vocabulaire 3 / 4

Clause

- Élément de la base de connaissance
- Affirmation inconditionnelle (ou fait) : un seul atome logique qui se termine par un point ≪ . ≫
- Affirmation conditionnnelle (ou règle) : succession d'atomes logiques tel que le premier (le but) et le deuxième sont séparés par `:-` et le deuxième et les autres (les prémisses) par `,` qui se termine par `.`
  La virgule lie les atomes par un ≪ et ≫ logique

- Il est à noter que `:-` et la virgule (`,`) sont des termes complexes d'arité 2 et le point (`.`) est un terme simple

Précisions concernant la syntaxe et le vocabulaire 4 / 4

Prédicats

- Ensemble de clauses dont leurs premiers atomes logiques (leurs buts) ont le même foncteur et la même arité
- Les clauses d'un même prédicat sont liées par un « ou » logique

Programme Prolog

- Ensemble de prédicats
- L'exécution d'un programme revient à poser une question

INSA ROUEN NORMANDIE

Les termes complexes 1 / 3

- Les termes complexes (qui ne sont pas des atomes logiques) servent à structurer hiérarchiquement les données

Attention

- Ils ne calculent rien, ce ne sont pas des fonctions

Exemple (inspiré de [?])

- Comment exprimer :
  - Annie possède une ford escort
  - Jérome possède une renault twingo
  - Jérome possède une console de jeu Sony playstation 5
  - Jérome possède une console de jeu Microsoft Xbox One
  - Hélène possède une renault mégane
  - Hélène possède une console de jeu switch

Les termes complexes 2 / 3

Avec uniquement des prédicats

```
possede(annie,voiture_annie).
possede(jerome,voiture_jerome).
possede(jerome,console_jerome_1).
possede(jerome,console_jerome_2).
possede(helene,voiture_helene).
possede(helene,console_helene).

marque(voiture_annie,ford).
marque(voiture_jerome,renault).
marque(console_jerome_1,sony).
marque(console_jerome_2,microsoft).
marque(voiture_helene,renault).
marque(console_helene,nintendo).

type_d_objet(voiture_annie,voiture).
type_d_objet(voiture_jerome,voiture).
type_d_objet(voiture_helene,voiture).
type_d_objet(console_jerome_1,console).
type_d_objet(console_jerome_2,console).
type_d_objet(console_helene,console).

modele(voiture_annie,escrot).
modele(voiture_jerome,twingo).
modele(console_jerome_1,playstation_5).
modele(console_jerome_2,xBox_One).
modele(voiture_helene,megane).
modele(console_helene,switch).
```

Qui possède une voiture de marque ford ?

```
?- possede(Personne,Objet),type_d_objet(Objet,voiture),marque(Objet,ford).
Personne = annie,
Objet = voiture_annie ;
false.
```

Les termes complexes 3 / 3

Avec les termes complexes (possede.pl)

```
possede(annie,voiture(ford,escort)).
possede(jerome,console(sony,playstation_3)).
possede(jerome,console(microsoft,xbox_one)).
possede(jerome,voiture(renault,twingo)).
possede(helene,console(nintendo,switch)).
possede(helene,voiture(renault,megane)).
```

Exemple d'interaction

```
1 ?- [possede].
true.

2 ?- possede(X,voiture(_,_)).
X = annie ;
X = jerome ;
X = helene.

3 ?- possede(X,voiture(renault,_)).
X = jerome ;
X = helene.
```

Les opérateurs

Du sucre syntaxique

- Un opérateur permet de représenter de manière plus pratique des termes complexes ou des prédicats d'arité 1 ou 2, par exemple :
  `op_g operateur op_d` correspond à `operateur(op_g,op_d)`
- Un opérateur est caractérisé par
  - une priorité : par exemple si l'opérateur `op1` et prioritaire à l'opérateur `op2`,
    `op_g op1 op_m op2 op_d` est interprété comme `op2(op1(op_g,op_m),op_d)`
  - une associativité (à gauche ou à droite) : par exemple si `op` est associatif à gauche,
    `op_g op op_m op op_d` est interprété comme `op(op(op_g,op_m),op_d)`

- Il est à noter que `:-` et `,` sont des opérateurs d'arité deux
- Le prédicat *write_canonical*1 permet d'obtenir la représentation sous forme de termes toute expression Prolog

Les expressions arithmétiques

Des termes complexes

- La notation `2+3+4` correspond au terme complexe `+(+(2,3),4)`
- Pour évaluer une expression arithmétique on utilise l'opérateur `is` qui correspond au prédicat *is/2* :
  `X is 2+3` correspondant au prédicat `is(X,2+3)` qui est vrai lorsque `X` est associé à `5`
- Les opérateurs `=:=`, `=\=`, `<`, `>`, `=<`, `>=` évaluent les expressions arithmétiques (opérandes gauches et droites) avant de les comparer

INSA ROUEN NORMANDIE

Les listes

Définition

- Terme complexe `'[|]'/2` tel que le premier arguments (un terme) est la tête de liste et le second la queue de liste (une liste).
- La liste vide est notée `[]`
- Constats :
  - Une liste permet de représenter une suite de termes
  - Sa définition récursive amène à écrire des prédicats récursifs

Facilité syntaxique, deux notations

- Les éléments de la liste sont entourés de crochets et séparés par des virgules : `'[|]'(1, '[|]'(2, []))` peut être notée `[1,2]`
- Une liste est entourée de crochet dont le premier élément est séparé du reste de la liste (une liste) par une barre : `'[|]'(1, '[|]'(2, []))` peut être notée `[1|[2|[]]]`

Les différentes façons de désigner une liste

```
?- '[|]'(1,'[|]'(2,'[|]'(3, []))) = [1,2,3].
true.

?- [1|[2|[3|[]]]] = [1,2,3].
true.

?- [1|[2|[3]]] = [1,2,3].
true.

?- [1,2|[3]] = [1,2,3].
true.
```

Quelques prédicats sur les listes 1 / 4

membre/2

```
/* membre(Element,Liste) est vrai si Element est un element de Liste*/
membre(Element,[Element|_]).
membre(Element,[_|Liste]) :- membre(Element,Liste).
```

Exemple de questions

```
?- membre(b,[a,b,c]).
true ;
false.

?- membre(X,[a,b,c]).
X = a ;
X = b ;
X = c ;
false.

?- membre(a,[a,b,a,c]).
true ;
true ;
false.
```

```
?- membre(a,L).
L = [a|_4450] ;
L = [_4448, a|_4456] ;
L = [_4448, _4454, a|_4462] ;
L = [_4448, _4454, _4460, a|_4468] ;
L = [_4448, _4454, _4460, _4466, a|_4474] ;
```

Quelques prédicats sur les listes 2 / 4

supprimer/3

```
/* supprimer(Element,Liste1,Liste2) est vrai lorsque Liste2 possede tous les elements de Liste1 sauf Element */
supprimer(_,[],[]).
supprimer(Element,[Element|Queue],QueueSansElement) :- supprimer(Element,Queue,QueueSansElement).
supprimer(Element,[ElementDifferent|Queue],[ElementDifferent|QueueSansElement]) :- Element\=ElementDifferent,
     supprimer(Element,Queue,QueueSansElement).
```

Exemple de questions

```
?- supprimer(1,[1,2,1,3],L).
L = [2, 3] ;
false.

?- supprimer(X,[1,2,1,3],[2,3]).
X = 1 ;
false.

?- supprimer(2,L,[1,3]).
ERROR: Out of local stack
   Exception: (1,595,448) supprimer(2, _9572670, [1, 3]) ?
```

Quelques prédicats sur les listes 3 / 4

longueur/2

```
/* longueur(Liste,Longueur) est vrai lorsque Longueur est égal au nombre d'éléments de Liste */
longueur([],0).
longueur([_|Queue], Longueur) :- longueur(Queue, LongueurQueue), Longueur is LongueurQueue+1.
```

Exemple de questions

```
?- longueur([1,2,3],3).
true.
?- longueur([1,2,3],X).
X = 3.
```

INSA ROUEN NORMANDIE

Quelques prédicats sur les listes 4 / 4

scinder_liste/2

```
/* scinder_liste(Liste,PremierPartie,DeuxiemePartie) est vrai lorsque Liste est scindée en deux liste de même
     longueur (à plus ou moins un près) */
scinder_liste([],[],[]).
scinder_liste(Liste,PremierPartie,DeuxiemePartie) :- append(PremierPartie,DeuxiemePartie,Liste), longueur(
     PremierPartie,LongueurPremierePartie), longueur(DeuxiemePartie,LongueurDeuxiemePartie),
     LongueurPremierePartie =:= LongueurDeuxiemePartie.
scinder_liste(Liste,PremierPartie,DeuxiemePartie) :- append(PremierPartie,DeuxiemePartie,Liste), longueur(
     PremierPartie,LongueurPremierePartie), longueur(DeuxiemePartie,LongueurDeuxiemePartie),
     LongueurPremierePartie+1 =:= LongueurDeuxiemePartie.
```

Exemple de questions

```
?- scinder_liste([1,2,3,4],L1,L2).
L1 = [1, 2],
L2 = [3, 4] ;
false.

?- scinder_liste([1,2,3],L1,L2).
L1 = [1],
L2 = [2, 3] ;
false.
```

Conclusion

Ce qu'il faut retenir

- La programmation logique est un paradigme de programmation : le développeur décrit le **quoi** et non pas le **comment**
- Prolog est un langage de programmation logique, pour lequel nous avons :
  - présenté le contenu d'un programme : faits, règles et question
  - précisé le vocabulaire : termes simples (atome, constante, variable), terme complexe, atome logique, clause et prédicat
  - présenté en détail les termes complexes
  - présenté les opérateurs qui permettent de plus facilement utiliser certains termes complexes ou certains prédicats d'arité un ou deux
  - présenté les expressions arithmétiques qui sont des termes complexes par défaut non évaluées
  - présenté les listes et quelques algorithmes

INSA ROUEN NORMANDIE