Chapitre 6 : Bases de Prolog

Carole Porrier
carole.porrier@univ-paris13.fr

Département d'informatique
IUT de Villetaneuse

30 avril 2024
Paradigmes de développement universels

Table des matières

Table des matières

Syntaxe de base

- Un programme **Prolog** est construit à l'aide d'un ensemble de **termes**;
- Un **terme** est une **constante**, une **variable** ou une **structure**.
- Un **terme** est identifié par une suite de **caractères**.

Type	Exemples
Majuscule	A, B, C, ..., Z
Minuscule	a, b, c, ..., z
Chiffres	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Symboles	`+`, `-`, `*`, `/`, `\`, `~`, `^`, `<`, `>`, `:`, `.`, `?`, `@`, `#`, `$`, `&`, ...

Constantes (ou litéraux)

- Il existe deux types de **constantes**: les **atomes** et les **nombres**;
- Quelques atomes:

```
% Valides
heureuse  sont_freres  sontFreres
'Pomme'  'Avec des espaces'
?-  :0  -->  =

% Invalides
12pomme  Pomme  _nom
```

- Quelques nombres:

```
-17  -2.67  0  1  -4.311e4  6.02e-23
```

Variables

- Quelques variables:

```
Answer  _nom  X1  Y_3  _
```

- La variable **spéciale** _ (caractère de soulignement) indique qu'on **ne se soucie pas** de son contenu;
- Permet d'améliorer grandement la **lisibilité**.

Structures (termes composés)

- Permettent de **regrouper** des constantes de façon logique;
- Quelques termes composés:

```
point(4.0, 5.1)
segment(point(4.0, 5.1),point(-1.0,8.7))
livre('Les cerfs-volants', 'Romain Gary', 425)
```

- Essentiellement l'idée d'**enregistrement** (*record*, *struct*) dans les autres langages;
- Facilite l'écriture de **requêtes** et permet de mieux **représenter** les connaissances.

Prédicats et faits

- Un **fait** est un cas particulier de **structure**:

```
sont_freres(bob,carl).
travaille(alice,hopital,lundi).
travaille(bob,universite,jeudi).
```

- Un **prédicat** est une **structure** avec des **variables**:

```
sont_freres(X,Y) :-
    a_mere(X,M),
    a_mere(Y,M),
    a_pere(X,P),
    a_pere(Y,P).
```

Opérateurs

- En Prolog, les symboles `+`, `-`, `*` et `/` peuvent être traités comme des **atomes quelconques**;
- Ainsi, les deux **écritures** suivantes sont équivalentes:

```
+(x,*(y,z)) % Notation préfixe
x + y * z   % Notation infixe
```

- En particulier, `4+9` et `13` sont deux objets **différents**:

```
?- 4 + 9 = 13.
false.
?- +(4,9) = 13.
false.
```

- Par contre,

```
?- 4 + 3 = X + 2.
false.
?- 4 + 3 = X + 3.
X = 4.
```

Priorité des opérations

- Par défaut, les opérateurs ont une **priorité** prédéfinie;
- On peut utiliser des **parenthèses** pour court-circuiter la priorité.
- On verra plus tard qu'il est assez facile de **redéfinir** la priorité nous-mêmes;
- Très **pratique** si on utilise des **symboles** pour qu'ils aient un sens particulier.
- Autrement dit, en Prolog, il est facile de **redéfinir** les opérateurs.

Table des matières

L'opérateur d'égalité

- Lorsqu'on écrit `X = Y`, on cherche à vérifier s'il y a une façon de **rendre égales** les deux expressions;
- Si c'est possible, on dit que `X` et `Y` sont **unifiés**.
- L'**égalité** est définie comme suit:
  - On compare des **atomes** avec des **atomes**;
  - On compare des **entiers** avec des **entiers**;
  - On compare des **flottants** avec des **flottants**;
  - On compare des **structures** avec des **structures**;
- S'il y a des variables **non instantiées** et **compatibles**, le but **réussit**;
- Sinon, le but **échoue** (`fail`).

Relation d'égalité

- Deux **atomes** sont égaux si et seulement s'ils sont les **mêmes**;
- Deux **entiers** sont égaux si et seulement s'ils sont les **mêmes**;
- Deux **flottants** sont égaux si et seulement s'ils ont la même **valeur**.
- Deux **structures** sont égales si et seulement s'ils elles ont
  - le même **foncteur**;
  - le même nombre d'**arguments** et
  - toutes leurs composantes sont elles-même **égales** (c'est récursif).

Exercices

Dire si les buts suivants **réussissent** ou **échouent**:

```
pilots(A, london) = pilots(london, paris).
point(X, Y, Z) = point(X1, Y1, Z1).
letter(C) = word(letter).
noun(alpha) = alpha.
'student' = student.
f(X, X) = f(a, b).
f(X, a(b, c)) = f(Z, a(Z, c)).
```

(Les questions sont tirées du livre de référence.)

Table des matières

Logique et arithmétique

- À la base, Prolog a été conçu pour représenter des **raisonnements**;
- Et des notions de **logique**.
- En particulier, quand on modélise un problème, les traitements **logique** et **arithmétique** sont souvent séparés;
- Bien que Prolog propose des **mécanismes** arithmétiques, ils utilisent une syntaxe parfois **contre-intuitive**.

Comparaison de nombres

- On a les opérateurs **habituels**:

Opérateur	Description
`X =:= Y`	égalité de deux nombres
`X =\= Y`	non égalité de deux nombres
`X < Y`	strictement plus petit
`X > Y`	strictement plus grand
`X =< Y`	plus petit ou égal
`X >= Y`	plus grand ou égal

- Noter que `X = Y` et `X =:= Y` **n**'ont **pas** le même comportement:

```
?- 2 = 2.0.
false.
?- 2 =:= 2.0.
true.
```

Opérateurs arithmétiques

- On a les opérateurs **habituels**:

Opérateur	Description
`X + Y`	somme
`X - Y`	différence
`X * Y`	produit
`X / Y`	division
`X // Y`	division entière
`X mod Y`	reste de la division entière

- Des fonctions **mathématiques** sont aussi disponibles:

```
?- sqrt(2, X).
X = 1.414213562373095.
```

- Attention à la **syntaxe** qui peut surprendre. Le **prédicat** est vrai si X est la racine carrée de 2.

Évaluation

- Rappel: les expressions arithmétiques ne sont pas **évaluées** lorsqu'on utilise l'opérateur `=` :

```
?- 4 + 3 = X + 2.
false.
?- 4 + 3 = X + 3.
X = 4.
```

- Ce n'est évidemment **pas pratique** si on souhaite faire de l'arithmétique de **base**;

- En Prolog, pour **évaluer** une telle expression, on utilise le mot réservé `is`:

```
?- X is 4 + 7.
X = 11.
?- Y is 8 // 3.
Y = 2.
?- A is sqrt(2).
A = 1.4142135623730951.
```

Exercice

- Implémenter en Prolog un prédicat qui permet de calculer la **moyenne de deux nombres**;
- Implémenter en Prolog un prédicat qui permet de vérifier si un nombre est une **puissance de 2**;
- Implémenter en Prolog un prédicat qui indique si un point donné se trouve **à l'intérieur** d'une sphère donnée.

Table des matières

Structures habituelles

- Les **structures** ont naturellement une structure **arborescente**;
- Les **listes**, qui ont une structure **séquentielle**, sont aussi supportées.
- Pour les **associations**, on utilise plutôt les **prédicats** directement;
- Nous verrons plus tard dans le cours qu'il est possible de manipuler assez facilement les **graphes**.

Comparaison de structures (1/2)

- Il est possible de comparer des **structures** :

```
?- alpha @< beta.
true.
?- 'a' @< "a".
false.
?- "a" @< 'a'.
true.
?- alpha(beta) @< alpha(gamma).
true.
?- beta(2, 3) @< alpha(4).
false.
?- beta(2) @< alpha(4, 2).
true.
```

- Comment est définie la **relation d'ordre**?

Comparaison de structures (2/2)

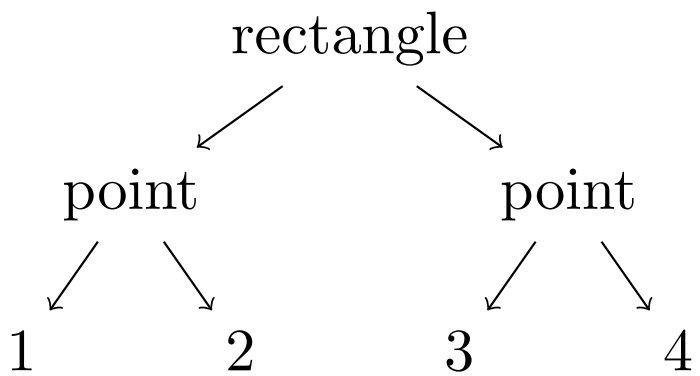
- Dans un premier temps, on compare les **arités** des structures;
- S'il y a égalité, alors on compare les **foncteurs**, selon l'ordre **lexicographique** (l'ordre ASCII);
- S'il y a égalité, alors on compare le **premier argument**;
- S'il y a égalité, alors on compare le **deuxième argument**;
- Ainsi de suite, jusqu'au **dernier argument**;
- On obtient ainsi une relation d'ordre **totale**.

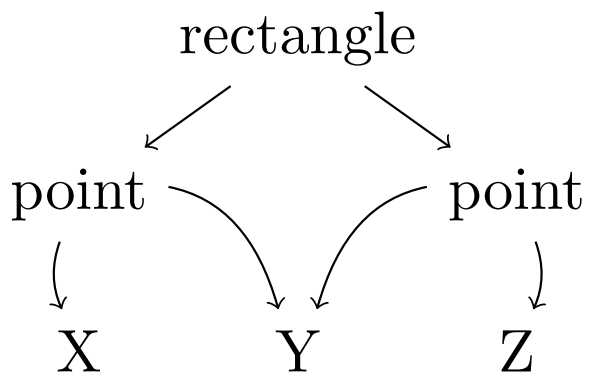
Structures, arbres et GOA

Une **structure** cache

- un **arbre** ou
- un **graphe orienté acyclique** (*directed acyclic graph ou DAG*).

```
% Deux structures
rectangle(point(1,2),point(3,4))
rectangle(point(X,Y),point(Y,Z))
```





Arbre binaire

- On peut exploiter l'**arborescence** d'une structure pour organiser l'information en **arbre binaire**;

```
% Par défaut, la recherche est séquentielle
cours('ECO1081', "Économie des technologies de l'information").
cours('INF1070', 'Utilisation et administration des sytèmes informatiques').
cours('INF1120', 'Programmation I').
cours('INF1131', 'Mathématiques pour informaticiens').
cours('INF2050', 'Outils et pratiques de développement logiciel').
cours('INF2120', 'Programmation II').
cours('INF2171', 'Organisation des ordinateurs et assembleur').
cours('INF3080', 'Bases de données').
cours('INF3105', 'Structures de données et algorithmes').


...

% Recherche dans un arbre binaire
titre(S, noeud(S, T, _, _), T1) :- !, T = T1.
titre(S, noeud(S1, _, G, _), T) :-
    S @< S1,
    titre(S, G, T).
titre(S, noeud(S1, _, _, D), T) :-
    S @> S1,
    titre(S, D, T).
```

Listes

- Comme dans **Haskell**, les listes sont disponibles en **Prolog**:

```
?- [X,Y,Z] = [1,2,3].
X = 1,
Y = 2,
Z = 3.
?- [X,Y] = [3,2,1].
false.
?- [X,Y] = [[1,2],[[3]]].
X = [1, 2],
Y = [[3]].
```

- On peut décomposer en **tête** et **queue** avec le symbole “pipe” (|) :

```
?- [X|Y] = [1,2,3,4,5].
X = 1,
Y = [2, 3, 4, 5].
```

Exercices

- Définir un prédicat

  `membre(X, L)`

  qui indique si `X` apparaît dans la liste `L`;

- Définir un prédicat

  `longueur(L, N)`

  qui indique si `L` est de longueur `N`.

- Définir un prédicat

  `palindrome(L)`

  qui indique si `L` est un palindrome.