Université Constantine 2
جامعة قسنطينة 2

Programmation Logique et Fonctionnel (PLF)

Chapitre 4 : Programmation PROLOG

Dr. Boudouda Souheila

Faculté:NTIC/Département: **TLSI**

Boudouda.souheila@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
NTIC	TLSI	Master 1	Génie Logiciel (GL)

Historique

Origine

- PROLOG : PROgrammation LOGique.
- 1967 : SIMULA 67
- 1972 : introduit par Alain COLMERAUER. Ce langage permet une programmation « déclarative ». Repose sur deux courants: la définition des langages de programmation (« W-grammaires ») et la démonstration automatique
- 1981 : programme Japonais de 5ème Génération.
- 1989 : Programmation Logique avec Contraintes (Prolog III)
- 1995 : Extension et introduction des contraintes sur intervalles (Prolog IV).
- …….

SWI-PROLOG

SWI-PROLOG

- Interpréteur PROLOG disponible gratuitement
- Fonctionne avec: Linux, Windows, ou Mac OS
- Plusieurs Interpréteurs PROLOG
- Ne sont pas tous ISO Standards
- http://www.swi-prolog.org/

Autres Logiciels Prolog

- Domaine du « libre »: INRIA : GNU-Prolog, Sicstus, Eclipse.
- Domaine payant: PrologIA : Prolog 4, COSYTEC (Orsay) : CHIP (Constraint Handling in Prolog), ILOG Solver.

...

Méthodologie de construction de programme

Caractéristique d'un programme PROLOG :

- Programmation Logique
- Programmation Déclarative
- Très différent des autres langages de programmation (procédurale)
- Approprié pour des tâches orientées connaissances

Méthodologie de construction de programme

- Spécification logique à partir de laquelle on dérive un programme à l'aide de « règles de dérivation ».
- Le langage dans lequel la spécification et le programme sont écrits est le « même ».
- Les règles de dérivation conservent la «logique » : si le programme s'arrête le résultat est conforme à la spécification.

Idée de Base de PROLOG

Comment construire un programme Prolog

- Décrire une situation donnée
- Poser des questions
- Prolog déduit logiquement de nouveaux faits autour d'une situation décrite
- Prolog nous retourne ses déductions comme réponses

Conséquences:

- Réflichir d'une manière déclarative, non procédurale
  - Défi
  - Nécessite une manière différente de raisonnement
- Langage de Haut Niveau
  - Bon pour le prototypage rapide
  - Utile dans beaucoup d'applications IA

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- femme(sara).
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- femme(sara).
yes
?-
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- femme(sara).
yes
?- playsAirGuitar(hind).
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- femme(sara).
yes
?- playsAirGuitar(hind).
yes
?-
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- femme(sara).
True
?- playsAirGuitar(hind).
True
?- playsAirGuitar(sara).
false
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- tattoed(hind).
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- tattoed(hind).
false
?-
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- tattoed(hind).
ERROR: predicate tattoed/1 not defined.
?-
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- party.
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- party.
True
?-
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- rockConcert.
```

Exemple initial type BC1 (Base de connaissance1)

```
femme(sara).
femme(hind).
femme(yamina).
playsAirGuitar(hind).
party.
```

```
?- rockConcert.
ERROR: predicate rockConcert not defined.
?-
```

Exemple BC2 (Base de connaissance2)

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

Exemple BC2 (Base de connaissance2)

Fait

happy(yamina).

Listens2music(sara).

listens2music(yamina):- happy(yamina).

playsAirGuitar(sara):- listens2music(sara).

playsAirGuitar(yamina):- listens2music(yamina).

Exemple BC2 (Base de connaissance2)

happy(yamina). Fait

Listens2music(sara). Fait

listens2music(yamina):- happy(yamina).

playsAirGuitar(sara):- listens2music(sara).

playsAirGuitar(yamina):- listens2music(yamina).

Exemple BC2 (Base de connaissance2)

happy(yamina).

Fait

Listens2music(sara).

Fait

listens2music(yamina):- happy(yamina).

Règle

playsAirGuitar(sara):- listens2music(sara).

playsAirGuitar(yamina):- listens2music(yamina).

Exemple BC2 (Base de connaissance2)

happy(yamina). Fait

listens2music(sara). Fait

listens2music(yamina):- happy(yamina). Règle

playsAirGuitar(sara):- listens2music(sara). Règle

playsAirGuitar(yamina):- listens2music(yamina).

Exemple BC2 (Base de connaissance2)

happy(yamina). Fait

Listens2music(sara). Fait

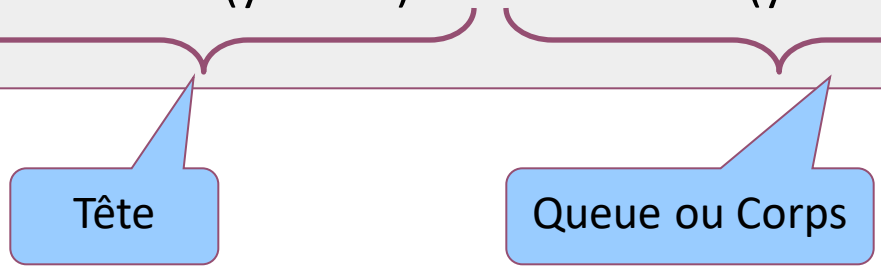
listens2music(yamina):- happy(yamina). Règle

playsAirGuitar(sara):- listens2music(sara). Règle

playsAirGuitar(yamina):- listens2music(yamina). Règle

Exemple BC2 (Base de connaissance2)

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```



Exemple BC2 (Base de connaissance2)

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

```
?-
```

Exemple BC2 (Base de connaissance2)

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

```
?- playsAirGuitar(sara).
True
?-
```

Exemple BC2 (Base de connaissance2)

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

```
?- playsAirGuitar(sara).
True
?- playsAirGuitar(yamina).
True
```

Clauses

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

*Il y'a 5 **clauses** dans cette Base de Connaissances:*
*deux **faits** et trois **règles**.*
*La fin d'une clause est marquée par un **point**.*

Les Prédicats

```
happy(yamina).
Listens2music(sara).
listens2music(yamina):- happy(yamina).
playsAirGuitar(sara):- listens2music(sara).
playsAirGuitar(yamina):- listens2music(yamina).
```

*Dans cette Base de Connaissances, il y'a **trois prédicats**:*
happy, listens2music, et playsAirGuitar

Exemple BC3

```
happy(farid).
Listens2music(hafedh).
playsAirGuitar(farid):- listens2music(farid), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh).
playsAirGuitar(hafedh):- listens2music(hafedh).
```

Expression Conjonctive

```
happy(farid).
Listens2music(hafedh).
playsAirGuitar(farid):- listens2music(faridt), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh).
playsAirGuitar(hafedh):- listens2music(hafedh).
```

La virgule "," traduit la conjonction en Prolog

Exemple BC3

```
happy(farid).
Listens2music(hafedh).
playsAirGuitar(farid):- listens2music(faridt), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh).
playsAirGuitar(hafedh):- listens2music(hafedh).
```

```
?- playsAirGuitar(farid).
false
?-
```

Exemple BC3

```
happy(farid).
Listens2music(hafedh).
playsAirGuitar(farid):- listens2music(faridt), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh).
playsAirGuitar(hafedh):- listens2music(hafedh).
```

```
?- playsAirGuitar(hafedh).
True
?-
```

Expressions Disjonctives

```
happy(farid).
Listens2music(hafedh).
playsAirGuitar(farid):- listens2music(faridt), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh).
playsAirGuitar(hafedh):- listens2music(hafedh).
```

```
happy(farid).
listens2music(hafedh).
playsAirGuitar(farid):- listens2music(farid), happy(farid).
playsAirGuitar(hafedh):- happy(hafedh); listens2music(hafedh).
```

Prolog et la Logique

- Prolog a un lien avec la Logique
- Opérations
  - Implication **:-**
  - Conjonction **,**
  - Disjonction **;**
- Utilisation du modus ponens
- Négation

Base de Connaissances 4

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

Variables Prolog

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
```

Instanciation de Variables

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
X=meriem
```

Demander des Alternatives

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
X=meriem;
```

Demander des Alternatives

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
X=meriem;
X=jamila
```

Demander des Alternatives

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
X=meriem;
X=jamila;
X=yosra
```

Demander des Alternatives

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- woman(X).
X=meriem;
X=jamila;
X=yosra;
no
```

Demander des Alternatives

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- respect(malek,X), woman(X).
```

Base de Connaissances 4

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- respect(malek,X), woman(X).
X=meriem

?-
```

Base de Connaissances 4

```
woman(meriem).
woman(jamila).
woman(yosra).
respect(youssef, meriem).
respect(malek, meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- respect(pumpkin,X), woman(X).
```

Base de Connaissances 4

```
woman(mia).
woman(jody).
woman(yolanda).
respect(vincent, mia).
respect(marsellus, mia).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
```

```
?- respect(pumpkin,X), woman(X).
false
?-
```

Base de Connaissances 5

```
respect(youssef,meriem).
respect(malek,meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
aime(X,Y):- respect(X,Z), respect(Y,Z).
```

Base de Connaissances 5

```
respect(youssef,meriem).
respect(malek,meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
aime(X,Y):- respect(X,Z), respect(Y,Z).
```

```
?- aime(malek,W).
```

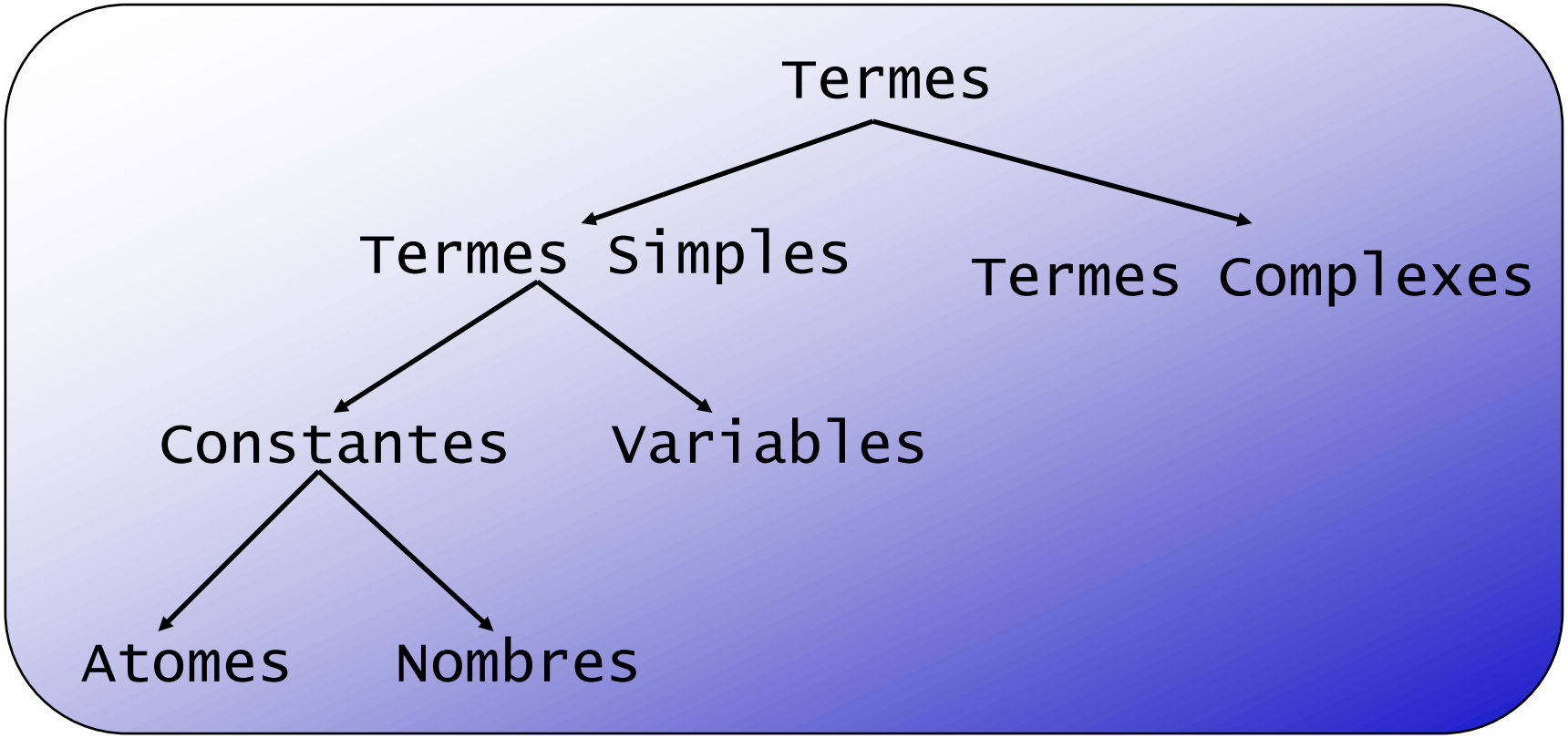
Base de Connaissances 5

```
respect(youssef,meriem).
respect(malek,meriem).
respect(pumpkin, honey_bunny).
respect(honey_bunny, pumpkin).
aime(X,Y):- respect(X,Z), respect(Y,Z).
```

```
?- aime(malek,W).
W=youssef
?-
```

Syntaxe de Prolog

- **Définitions exactes des faits, règles et requêtes?**



Atome

Atome:

- Séquence de caractères composée de lettres majuscules, de lettres minuscules, de chiffres ou de de souligné et commençant par une lettre minuscule
  - *exemples*: butch, big_kahuna_burger, playGuitar
- Toute séquence de caractères entre quotes
  - *Exemples*: ‘Youssef', 'Five dollar shake', '@$%'
- Une séquence de caractères spéciaux
  - *Exemples*: : , ; . :-

Nombre: Entiers : 12, -34, 22342, Réels: 34573.3234

Variables: Séquence de caractères composée de lettres majuscules, de lettres minuscules, de chiffres ou du souligné et commençant par soit une lettre majuscule ou du souligné

- Exemples: X, Y, Variable, Youssef, _tag

Termes Complexes

- Atomes, nombres et variables sont utilisés pour construire des termes complexes
- Les termes complexes sont construits à partir de fonctions directement suivies d'arguments
- Les arguments sont mis entre les crochets et séparés par des virgules
- Les fonctions doivent être des atomes
- **Exemples vus précedemment :**
  - playsAirGuitar(jamila)
  - respect(youssef, meriem)
  - aime(malek, W)
- **Termes Complexes à l'intérieur de termes complexes:**
  - hide(X,father(father(father(butch))))

Arité

- Le nombre d'arguments d'un terme complexe s'appelle son arité
- **Exemples:**
  woman(meriem) est un terme d'arité 1
  respect(youssef, meriem) est un terme d'arité 2
  father(father(butch)) est un terme d' arité 1
- En Prolog, on peut définir deux prédicats avec la même fonction et des arités différentes
- Prolog va les considérer comme deux prédicats différents
- En prolog, les arités des prédicats sont indiqués par un suffixe "/" suivi par le nombre d'arguments exple: respect/2.

Exemple d'arité

```
happy(yosra).
listens2music(meriem).
listens2music(yosra):- happy(yosra).
playsAirGuitar(meriem):- listens2music(meriem).
playsAirGuitar(yosra):- listens2music(yosra).
```

- **Cette Base de Connaissances définit**
  - happy/1
  - listens2music/1
  - playsAirGuitar/1

Unification et Stratégie de Recherche

- **Exemple :**
  - *frere(X,Y) :- homme(X), enfant(X,Z), enfant(Y,Z), X\=Y.* où \= représente le prédicat de différence.
  - *frere(Omar,Qui)* : tentative d'unification avec la tête de la clause *frere(X,Y)*
- **Définition :** procédé par lequel on essaie de rendre deux formules identiques en donnant des valeurs aux variables qu'elles contiennent.
- **Résultat :** c'est un unificateur (ou substitution), un ensemble d'affectations de variables.
  - Exemple : *{X=Omar, Qui=Y}*
- Le résultat n'est pas forcément unique, mais représente l'unificateur le plus général.
- L'unification peut réussir ou échouer.
  - *e(X,X)* et *e(2,3)* ne peuvent être unifiés.

Unification

- **Prédicat d'unification : « = »**
  - *a(B,C) = a(2,3).* donne pour résultat :

    YES {B=2, C=3}
  - *a(X,Y,L) = a(Y,2,Aicha).* donne pour résultat :

    YES {X=2, Y=2, L=Aicha}
  - *a(X,X,Y) = a(Y,u,v).* donne pour résultat :

    NO

Étapes de démonstration

- Si l'unification échoue : situation d'échec sur la règle considérée pour démontrer la formule.

- Si l'unification réussit : substitution des variables présentes dans la queue de la clause par les valeurs correspondantes des variables de l'unificateur.

- Démonstration de cet ensemble de formules dans l'ordre de leur citation pour enrichir le système avec les valeurs obtenues des variables.

- A la fin, l'ensemble des couples valeur-variable des variables présentes dans la question initiale forme la solution affichée par Prolog.

Exemple

- **Exemple:**

ancêtre (X, Y) :- père(X, Y).
ancêtre (X, Y) :- ancêtre(X, Z), père(Z, Y).
père(mouadh, soufiane).
ancêtre(omar, mouadh).

?- pére(mouadh, soufiane).

true

?- pére (soufiane, mouadh).

false

pére(X, soufiane).

X=mouadh

pére(X, Y).

X= mouadh,

Y= soufiane

Exemple

- ?- mouadh = mouadh

true

- ?- mouadh = soufiane

false

- ?- X = mouadh
- X = mouadh

?- mouadh = X.

X = mouadh

?- pére (X, soufiane) = pere(mouadh, soufiane).

X=mouadh

Exemple

?- pére (X, Z) = pére (mouadh, Y).

X = mouadh

Z = Y

?- ((mouadh, soufiane), omar) = (X, Y), X =(M, N).

X = mouadh, soufiane,

Y= omar,

M = mouadh,

N = Soufiane

L'unification est un mecanisme interne de Prolog

Résumé

- Des exemples simples de programmes Prolog
- Introduction des trois structures de base en Prolog:
  - Faits
  - Règles
  - Requêtes
- Discussion des autres concepts tels que
  - Le rôle de la Logique
  - unification avec l'aide des variables
- Définition des constructions en Prolog:
  termes, atomes, et variables

Les Listes en Prolog

- Séquence finie d'éléments
- [m1, m2, m3, m4]
- [m1, r(h), X, 2, m1]
- []
- [[], d(ali), [2, [b,c]], [], [2, [b, c]]]
- La longueur d'une liste est le nombre d'élements
- **Head, Tail:**
- Toute liste non vide est constituée de deux parties:
- Head (tête de liste): le 1er element de la liste
- Tail (queue de la liste): ce qui reste aprés avoir enlever le premier (la tête)
- La queue d'une liste est toujours une liste

Head et Tail

- **Head et Tail: exemple 1**
- [meriem, farid, yasser, yasmina]
  Head: meriem
  Tail: [farid, yasser, yasmina]
- **Head et Tail: exemple 2**
- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
  Head: [ ]
  Tail: [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]
- **Head et Tail: exemple 3**
- Head: dead(z)
  Tail: [ ]
- **Head et tail d'une liste vide**
- La liste vide n'a ni Head ni Tail
- Pour Prolog, [ ] est une liste speciale qui n'a aucune structure interne

Décomposition d'une liste

Opérateur | Permet de décomposer une liste en tête et queue

?- [Head | Tail] = [m1, m2, m3, m4].

Head = m1

Tail = [m2, m3, m4]

Yes

?- [X | Y] = [m1, m2, m3, m4].

Décomposition d'une liste

- **1- Representation des Listes :**
- Écrire des listes
- La liste vide []
- L'opérateur | (extraction des informations à partir des listes
- Head et Tail (H et T) sont des variables qui expriment la tête (premier élément de la liste) et la queue (une liste sauf le premier élément)
- **Manipuler les listes:**
- Exprimer des requêtes

?- [Head | Tail] = [m, v, j, y]

?- [X | Y] = [m, v, j, y]

?- [X | Y] = []

?- [X | Y] = [[], dead(zed), [2, [b, c]], [], Z].

?- [X, Y | W] = [[], dead(zed), [2, [b, c]], [], Z].

Décomposition d'une liste

- La liste [[], dead(zed), [2, [b, c]], [], Z].
- ?- [X1, X2, X3, X4 | Tail] = [[], dead(zed), [2, [b, c]], [], Z].
- ?- [_, X, _, Y |_] = [[], dead(zed), [2, [b, c]], [], Z].
- Les variables anonymes ne sont pas visualisées par Prolog
- La liste interne [2, [b, c]] extraire la tête de cette liste interne a partir de la liste:

[[], dead(zed), [2, [b, c]], [], Z].

?- [_, X, _, Y | _] = [[], dead(zed), [2, [b, c]], [], Z].

?- [_, _,[_|X] | _] = [[], dead(zed), [2, [b, c]], [], Z].

Les opérations arithmétiques

Arithmétique	PROLOG
6 + 2 = 8	8 is 6 + 2
6 * 2 = 12	12 is 6 * 2
6 – 2 = 4	4 is 6 - 2
6 – 8 = -2	-2 is 6 - 8
6 / 2 = 3	3 is 6 / 2
7 / 2 = 3	3 is 7 / 2
Mod (7, 2) = 1	1 is mod(7, 2)

Exemple

?- 8 is 6 + 2.

Yes

?- 12 is 6 * 2.

Yes

?- -2 is 6 – 8.

Yes

?- 1 is mod(7, 2).

yes

Exemple

?- X is 6 + 2.

X = 8

?- R is mod(7, 2).

R = 1

add_3_and_double (X, Y) :- Y is (X+3)*2.

?- add_3_and_double(1, X).

X = 8

?- add_3_and_double(2, X).

X = 10

Priorités des operateurs

- Prolog comprend et respecte la règles des priorités des operateurs. Il resoud lui même les ambigüités des opérations.

?- X is 3 + 2 * 4.

X = 11

- Prolog n'effectue pas des opérations arithmétiques

?- X = 3 + 2.

X = 3 + 2

Yes

- Prolog ne fait que correspondre la variable X au terme complexe 3 + 2. Il n'effectue aucune operation arithmetique. Il ne fait que l'unification.
- ?- 3 + 2 * 5 = X
- X = 3 + 2 * 5
- Yes

Priorités des opérateurs

- Pour forcer l'evaluation, on doit utiliser is (les operations arithmetiques sont extra).
- Normalement Prolog ne fait que le Matching.
- **Restrictions sur les opérations:**
- L'expression arithmétique doit être après "is"

?- 6 + 2 is X est une erreur (erreur d'instanciation)

- Lorsque on effectue une opération, les variables doivent être déjà instanciées à leurs types (entiers).
- Prolog fait appel à une boite noire performant les opérations arithmétiques et doit donc respecter les conditions de types sinon on a une erreur d'instanciation

Exemple

Le prédicat add_3_and_double(X, Y) :

add_3_and_double (X, Y) :- Y is (X+3)*2.

add_3_and_double (3,Y) retourne Y= 12.

L'utilisation de ce prédicat dans le sens contraire(évaluer X étant donne Y) nous donne erreur d'instanciation

?- add_3_and_double (X, 12). erreur d'instanciation

?- X is + (3, 2).

X = 5

?- is (X, +(3, 2)).

X = 5

Récursivité et Listes

- Ecrire une fonction Member pour déterminer si un élément appartient à une liste

```
member(X, [X | T]) .

member (X, [H | T]) :- member(X, T).
```

- Autre écriture de member

```
member(X, [X | _]) .

member (X, [_ | T]) :- member(X, T).
```

Exemples

- **Longueur d'une liste:**

```
len ([], 0).
len ([_ | T], N) :- len (T, X), N is X + 1.

?- len ([a, b, c, d, e, [a, b], g], X).
X = 7
```

- **Concatenation des deux listes:**
- concat ([], L, L).
- concat ([H | T], L2, [H | L3]) :- concat(T, L2, L3).

Comparaison des entiers

X < Y.

X =< Y.

X =:= Y.

X =\= Y.

X > Y.

X >= Y

?- 2 < 4.

Yes

?- 4 =< 4.

Yes

?- 2 + 1 < 4.

Yes

?- X < 3.

Erreur

?- 3 < Y

Erreur

?- X =:= X.

Erreur

?- X = 3, X < 4.

Yes

?- X = b, X < 4.

erreur

Comparaison des Nombres

```
AccMax ([H|T], A, Max) :- H > A, accMax (T, H, Max).
AccMax ([H|T], A, Max) :- H =< A, accMax (T, A, Max).
accMax([], A, A).
```