Programmation Logique

Arithmétique

Enseignant: NGUYEN Thi Minh Tuyen

Arithmétique en Prolog

- Prolog fournit quelques opérateurs arithmétiques sur nombres entiers et réels.
- Affectation: `is`
- Fonctions prédéfinies :
  - `-`, `+`, `*`, `^`,
  - Division entière (symbole `//`), division flottante (symbole `/`)
  - `mod`, `abs`, `min`, `max`, `sign`, `random`, `sqrt`, `sin`, `cos`, `tan`, `log`, `exp`, ln, ...

Exemples

Arithmétique	Prolog
2 + 3 = 5	
2 x 3 = 6	
4 – 2 = 2	
2– 4 = –2	
4 : 2 = 2	
1 est le reste de la division de 7 et 2	

Exemples


```
?-X is 10.5+4.7*2.
X = 19.9
?- Y is 10, Z is Y+1.
Y = 10, Z = 11
?- X is sqrt(36).
X=6
?- X is 10,Y is -X-2.
X = 10, Y=-12
```

X+Y	The sum of X and Y
X-Y	The difference of X and Y
X*Y	The product of X and Y
X/Y	The quotient of X and Y
X//Y	The 'integer quotient' of X and Y (the result is truncated to the nearest integer between it and zero)
X mod Y	The remainder when X is divided by Y
X^Y	The value of X to the power of Y
-X	The negative of X
abs(X)	The absolute value of X
sin(X)	The sine of X (for X measured in radians)
cos(X)	The cosine of X (for X measured in radians)
max(X,Y)	The larger of X and Y
round(X)	The value of X rounded to the nearest integer
sqrt(X)	The square root of X

Unification

- Deuxième argument de l'opérateur `is/2` est évalué et puis cette valeur est unifiée avec le premier argument.
  - Si le premier argument est une variable non liée, il est lié à la valeur du deuxième argument et le but `is` réussi.
  - Si le premier argument est un nombre ou une variable liée avec une valeur numérique, il est comparé à la valeur du deuxième argument. S'ils sont identiques, le but `is` réussi, sinon il échoue.

Définition des prédicats avec arithétique

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

- Requête:

```
?- calculer(1,X).
X=8
?- calculer(2,X).
X=10
```

Arithmétique et liste

- La longueur d'une liste vide est 0.
- La longueur d'une liste non vide `[T|Q]`: 1 + `longueur(Q)` dans laquelle `longueur(Q)` est la longueur de la queue de la liste.
- En Prolog:

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

Accumulateur

- Nous utilisons l'accumalateur pour calculer la longeur d'une liste: `accLongeur(Liste,Acc,Longeur)`.
- En Prolog:

```
accLongeur([],A,A).
accLongeur([_|Q],A,L)  :-
    Anew  is  A+1,  accLongeur(Q,Anew,L).

longeur(Liste,Longeur) :-
  accLongeur(Liste,0,Longeur).
```

Comparaisons

Arithmétique	Prolog
$x < y$	
$x \leq y$	
$x = y$	
$x \neq y$	
$x \geq y$	
$x > y$	

Comparer deux termes

- T_1=:=T_2 réussit si T_1 est identique à T_2
- T_1=\=T_2 réussit si T_1 n'est pas identique à T_2
- T_1=T_2 unifie T_1 avec T_2
- T_1\=T_2 réussit si T_1 n'est pas unifiable à T_2
- Exemples :

```
?- p(A)=\=p(1).
true.
?- p(A)\=p(1).
false.
```

Exercices

- Définir les prédicats suivants en deux cas: sans accumulateur et avec accumulateur.

1. `min`
2. `max`

Exercices [1]

Définir les prédicats suivants:

1. `scalarMult/3` : prends trois arguments (un entier, une liste des entier, un liste résultante). Exemple:

   ?- scalarMult(3,[2,7,4],Result).

   Result = [6,21,12]

2. `dotProduct/3`: prends en arguments deux listes de même taille. Trosième arguments sera un entier qui est le produit scalaire de deux listes. Exemple:

   ```
   ?- dotProduct([2,5,6],[3,4,1],Result).
   Result = 32
   ```

3. `puissant/3` qui prends deux entiers x et y et retourne dans troisième argument le résultat de x^y

Exercices [2]

3. `fibonacci(N,R)` qui retourne l'entier à la position N de la suite de Fibonacci. Exemple:

   ```
   ?-fibonacci(3,R).
   R = 2
   ```

4. `Somme_des_impairs (L, S)` qui retourne dans S la sommes des nombres impairs dans la liste entrée L.
5. `Somme_des_pairs(L, S)` qui retourne dans S la sommes des nombres pairs dans la liste entrée L.
6. `Nombre_des_pairs(L, N)` qui retourne dans le nombre des nombres pairs dans la liste entrée L.

Opérateur

Opérateurs

- Prolog nous permet d'utiliser des notions des opérateurs plus conviviales.
- Par exemple: Expression arithmétique 2+2 signifie `+(2,2)`.
- Prolog fournit également d'un mécanisme pour ajouter notre opérateurs.

Propriétés des opérateurs

- Opérateur infixé
  - Les foncteurs sont écrits entre leur arguments
  - Exemple: `+  -  =  ==  -->`
  - Exemple: `likes(john,mary)` → `john likes mary`
- Opérateur préfixé
  - Les foncteurs sont écrits avant leur arguments
  - Exemple: négation –.
  - Exemple: `isa_dog(fred)` → `isa_dog fred`
- Opérateur postfixé
  - Les foncteurs sont écrits après leur arguments
  - Exemple: ++, –– dans la programmation en C
  - `fred isa_dog`

Priorité

- Chaque opérateur a une priorité afin d'éliminer ambiguïté.
- Exemple:
  - Est-ce que 2+3∗3 est équivalent à 2+(3∗3) ou (2+3)∗3 ?
  - Parce que la multiplication ∗ est plus prioritaire l'addition + ➔ 2+3∗3 est équivalent à 2+(3∗3).

Associativité

- Prolog utilise l'associativité pour désambiguïser les opérateurs ayant le même niveau de priorité.
- Exemple:
  - Est-ce que 2+3+4 est équivalent à `(2+3)+4` ou `2+(3+4)` ?
  - associatif à droite
  - associatif à gauche
- Des opérateurs peuvent également être définis comme non associatifs. Nous devons d'utiliser des parenthèses dans les cas ambigus.
  - Exemple: `:-` `-->`

Définir des opérateurs

- La définition d'un opérateur:
  - `:-op(Priorité, Type, Nom)`
  - Nom: nom de l'opérateur
  - Priorité : comprise entre 0 et 1200
  - Type: type de l'opérateur (infixé, associatif, etc.)

Types des opétateurs en Prolog

- `yfx` : opérateur infixé associatif à gauche
- `xfy` : opérateur infixé associatif à droite
- `xfx` : opérateur infixé non associatif
- `fx` : opérateur préfixé non associatif
- `fy` : opérateur préfixé associatif à droite
- `xf` : opérateur postfixé non associatif
- `yf` : opérateur postfixé associatif à gauche

1200	xfx	`-->`, `:-`
1200	fx	`:-`, `?-`
1150	fx	**dynamic**, **discontiguous**, **initialization**, **meta_predicate**, **module_transparent**, **multifile**, **public**, **thread_local**, **thread_initialization**, **volatile**
1100	xfy	`;`, `|`
1050	xfy	`->`, `*->`
1000	xfy	`,`
990	xfx	**:=**
900	fy	`\+`
700	xfx	`<`, `=`, `=..`, `=@=`, `\=@=`, `=:=`, `=<`, `==`, `=\=`, `>`, `>=`, `@<`, `@=<`, `@>`, `@>=`, `\=`, `\==`, **as**, **is**, `>:<`, `:<`
600	xfy	`:`
500	yfx	`+`, `-`, `/\`, `\/`, **xor**
500	fx	`?`
400	yfx	`*`, `/`, `//`, **div**, **rdiv**, `<<`, `>>`, **mod**, **rem**
200	xfx	`**`
200	xfy	`^`
200	fy	`+`, `-`, `\`
100	yfx	`.`
1	fx	`$`

Exemple


```
:-op(150,xfy,likes).
:-op(150,xf,is_female).
:-op(150,xf,isa_cat).
:-op(150,xfy,owns).

john likes X:- X is_female, X owns Y, Y isa_cat.
is_female(mary).
owns(mary,fido).
isa_cat(fido).
```

Question?