

Cours 5 : Arithmétique

- Contenu
 - Présentation des fonctionnalités de Prolog pour **l'arithmétique**
 - Application de ces fonctionnalités sur le traitement de listes simples, en utilisant des **accumulateurs**
- Exercices
 - Exercices de LPN: 5.1, 5.2, 5.3
 - Travaux pratiques

Arithmétique en Prolog

- Prolog fournit un ensemble d'outils pour l'arithmétique
- Entiers et nombre réels

Arithmétique

$2 + 3 = 5$
 $3 \times 4 = 12$
 $5 - 3 = 2$
 $3 - 5 = -2$
 $4 : 2 = 2$
1 est le reste de 7 divisé par 2

Prolog

?- 5 is 2+3.
?- 12 is 3*4.
?- 2 is 5-3.
?- -2 is 3-5.
?- 2 is 4/2.
?- 1 is mod(7,2).

Examples

?- 10 is 5+5.

yes

?- 4 is 2+3.

no

?- X is 3 * 4.

X=12

yes

?- R is mod(7,2).

R=1

yes

Définir un prédicat avec de l'arithmétique

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

Définir un prédicat avec de l'arithmétique

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

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

De plus près

- Il est important de savoir que $+$, $-$, $/$ et $*$ ne font pas d'arithmétique
- Des expressions comme $3+2$, $4-7$, $5/5$ sont des termes complexes
 - Foncteurs : $+$, $-$, $/$, $*$
 - Arity: 2
 - Arguments : entiers

De plus près

$$?- X = 3 + 2.$$

De plus près

?- $X = 3 + 2$.

$X = 3 + 2$

yes

?-

De plus près

?- $X = 3 + 2$.

$X = 3 + 2$

yes

?- $3 + 2 = X$.

De plus près

?- $X = 3 + 2$.

$X = 3 + 2$

yes

?- $3 + 2 = X$.

$X = 3 + 2$

yes

?-

Le prédicat is/2

- Pour forcer Prolog à évaluer des expressions arithmétique, il faut utiliser

is

- Il s'agit d'une instruction pour que Prolog effectue les calculs
- Comme il ne s'agit pas d'un prédicat ordinaire, il y a quelques restrictions d'utilisation

Le prédicat is/2

?- X is 3 + 2.

X = 5

yes

?- 3 + 2 is X.

ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.

Le prédicat is/2

?- X is 3 + 2.

Le prédicat is/2

?- X is 3 + 2.

X = 5

yes

?-

Le prédicat is/2

?- X is 3 + 2.

X = 5

yes

?- 3 + 2 is X.

Le prédicat is/2

?- X is 3 + 2.

X = 5

yes

?- 3 + 2 is X.

ERROR: is/2: Arguments are not sufficiently instantiated

?-

Le prédicat is/2

?- X is 3 + 2.

X = 5

yes

?- 3 + 2 is X.

ERROR: is/2: Arguments are not sufficiently instantiated

?- Result is 2+2+2+2+2.

Result = 10

yes

?-

Restrictions sur l'utilisation de `is/2`

- Il est possible d'utiliser des variables sur la droite du prédicat `is`
- Mais au moment où Prolog réalise l'évaluation, ces variables doivent être unifiées avec un terme Prolog qui n'est pas une variable
- Ce terme Prolog doit être une expression arithmétique

Notation

- Deux dernières remarques sur les expressions arithmétiques
 - $3+2$, $4/2$, $4-5$ sont des termes complexes représentées sous une forme facilement lisible :
 $3+2$ est en fait **$+(3,2)$** et ainsi de suite

Notation

- Deux dernières remarques sur les expressions arithmétiques
 - $3+2$, $4/2$, $4-5$ sont des termes complexes représentées sous une forme facilement lisible :
 $3+2$ est en fait **$+(3,2)$** et ainsi de suite

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

```
X = 5
```

```
yes
```

Arithmétique et Listes

- Quelle est la longueur d'une liste?
 - Longueur de la liste vide : zéro;
 - Longueur d'une liste non vide : un plus la longueur de sa queue.

Longueur d'une liste en Prolog

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

?-

Longueur d'une liste en Prolog

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

```
?- len([a,b,c,d,e,[a,x],t],X).
```

Longueur d'une liste en Prolog

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

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


Accumulateurs

- Ce programme est très bien :
 - Facile à comprendre
 - Relativement efficace
- Mais il existe une autre manière de calculer la longueur d'une liste
 - Présentons le principe des accumulateurs
 - Les accumulateurs sont des variables qui contiennent des résultats intermédiaires

Définissons acclen/3

- Le prédicat acclen/3 a trois arguments
 - La liste dont on veut calculer la longueur
 - La longueur de la liste, un entier
 - Un accumulateur, qui conserve la trace des valeurs intermédiaires pour le calcul de la longueur

Définissons acclen/3

- L'accumulateur de acclen/3
 - La valeur initiale de l'accumulateur est 0
 - On ajoute 1 à l'accumulateur chaque fois qu'il est possible de traiter la tête d'une liste
 - Quand nous atteignons la liste vide, l'accumulateur contient la longueur de la liste

Longueur d'une liste en Prolog

```
acclen([],Acc,Length):-  
    Length = Acc.
```

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

?-

Longueur d'une liste en Prolog

```
acclen([],Acc,Length):-  
    Length = Acc.
```

Ajoute 1 à
l'accumulateur chaque
fois que l'on atteint
la tête d'une liste

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

?-

Longueur d'une liste en Prolog

```
acclen([],Acc,Length):-  
    Length = Acc.
```

Quand on atteint la
liste vide,
l'accumulateur contient
la longueur de la liste

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

?-

Longueur d'une liste en Prolog

```
acclen([],Acc,Acc).
```

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

```
?-
```

Longueur d'une liste en Prolog

```
acclen([],Acc,Acc).
```

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

```
?-acclen([a,b,c],0,Len).
```

```
Len=3
```

```
yes
```

```
?-
```


Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).
/ \

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).

/

\

no

?- acclen([b,c],1,Len).

/

\

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).

/
no

\
?- acclen([b,c],1,Len).

/
no

\
?- acclen([c],2,Len).

/ \

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).

/
no

\
?- acclen([b,c],1,Len).

/
no

\
?- acclen([c],2,Len).

/
no

\
?- acclen([],3,Len).

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour acclen/3

?- acclen([a,b,c],0,Len).

/
no

\
?- acclen([b,c],1,Len).

/
no

\
?- acclen([c],2,Len).

/
no

\
?- acclen([],3,Len).

/
Len=3

\
no

```
acclen([ ],Acc,Acc).
```

```
acclen([_|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Ajouter un prédicat pour empaqueter (wrapper)

```
acclen([ ],Acc,Acc).
```

```
acclen([ _|L],OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

```
length(List,Length):-  
    acclen(List,0,Length).
```

```
?-length([a,b,c], X).
```

```
X=3
```

```
yes
```

Récurtivité terminal

- Pourquoi $\text{acclen}/3$ est meilleur que $\text{len}/2$?
 - $\text{acclen}/3$ est récursif terminal, pas $\text{len}/2$
- Différences:
 - Pour les prédicats récursifs terminaux, les résultats sont complètement calculé lorsque la fin de la récursion est atteinte
 - Pour prédicats récursifs qui ne sont pas terminaux, il reste des buts à réaliser quand la fin de récursivité est atteinte

Comparaison

Non récursif terminal

```
len([],0).  
len(_|L,NewLength):-  
    len(L,Length),  
    NewLength is Length + 1.
```

Récursif terminal

```
acclen([],Acc,Acc).  
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Arbre de recherche pour len/2

?- len([a,b,c], Len).

```
len([],0).  
len(_|L,NewLength):-  
    len(L,Length),  
    NewLength is Length + 1.
```

Arbre de recherche pour len/2

?- len([a,b,c], Len).

/ \

no ?- len([b,c],Len1),
 Len is Len1 + 1.

len([],0).

len([_|L],NewLength):-

 len(L,Length),

 NewLength is Length + 1.

Arbre de recherche pour len/2

?- len([a,b,c], Len).

/ \

no ?- len([b,c],Len1),
Len is Len1 + 1.

/ \

no ?- len([c], Len2),
Len1 is Len2+1,
Len is Len1+1.

len([],0).

len([_|L],NewLength):-
len(L,Length),
NewLength is Length + 1.

Arbre de recherche pour len/2

?- len([a,b,c], Len).

/ \

no ?- len([b,c], Len1),
Len is Len1 + 1.

/ \

no ?- len([c], Len2),
Len1 is Len2+1,
Len is Len1+1.

/ \

no ?- len([], Len3),
Len2 is Len3+1,
Len1 is Len2+1,
Len is Len1 + 1.

len([],0).

len(_|L, NewLength):-

len(L, Length),

NewLength is Length + 1.

Arbre de recherche pour len/2

?- len([a,b,c], Len).

/ \

no ?- len([b,c], Len1),
Len is Len1 + 1.

/ \

no ?- len([c], Len2),
Len1 is Len2+1,
Len is Len1+1.

/ \

no ?- len([], Len3),
Len2 is Len3+1,
Len1 is Len2+1,
Len is Len1 + 1.

/ \

Len3=0, Len2=1,
Len1=2, Len=3

len([],0).

len([_|L], NewLength):-

len(L, Length),

NewLength is Length + 1.

Arbre de recherche pour len/2

?- acclen([a,b,c],0,Len).

/
no

\
?- acclen([b,c],1,Len).

/
no

\
?- acclen([c],2,Len).

/
no

\
?- acclen([],3,Len).

/
Len=3

\
no

```
acclen([ ],Acc,Acc).
```

```
acclen(_|L,OldAcc,Length):-  
    NewAcc is OldAcc + 1,  
    acclen(L,NewAcc,Length).
```

Exercises

- Exercise 5.1
- Exercise 5.2
- Exercise 5.3

Comparaison d'entiers

- Certains prédicats arithmétiques font des opérations eux-même, sans is/2
- Ce sont des opérateurs qui comparent des entiers

Comparaison d'entiers

Arithmetic

$x < y$

$x \leq y$

$x = y$

$x \neq y$

$x \geq y$

$x > y$

Prolog

$X < Y$

$X = < Y$

$X =: = Y$

$X = \backslash = Y$

$X > = Y$

$X > Y$

Comparaison d'entiers

- Force l'évaluation des arguments de gauche et de droite

?- $2 < 4+1$.

yes

?- $4+3 > 5+5$.

no

Comparaison d'entiers

- Force l'évaluation des arguments de gauche et de droite

?- 4 = 4.

yes

?- 2+2 = 4.

no

?- 2+2 =:= 4.

yes