

# Programmation Logique – TP0

N. Jussien (sur une idée de R. Debruyne)

2001 – 2002

## 1. Arithmétique réversible en Prolog

On choisit de représenter en Prolog les entiers naturels par `zero` et ses successeurs. Ainsi, 1 est représenté par `s(zero)` et 3 par `s(s(s(zero)))`.

- a. On définit le prédicat `is_nat/1` qui reconnaît un tel entier.

```
is_nat(zero).  
is_nat(s(X)) :- is_nat(X).
```

On pose la question `is_nat(Num)`. Que se passe-t-il ? Expliquer.

On inverse l'ordre des deux classes. Que se passe-t-il ? Expliquer.

- b. Que fait le prédicat `foo/2` ?

```
foo(0, zero).  
foo(N, s(X)) :- N1 is N-1, foo(N1, X).
```

- c. Définir le prédicat `oof/2` qui fait le traitement inverse.

- d. Pourquoi a-t-on besoin de deux versions spécialisées ?

- e. Définir le prédicat `somme/3` qui réalise la somme de deux entiers ainsi représentés :

```
?- somme(s(s(zero)), s(s(s(zero))), X).
```

```
    X = s(s(s(s(s(zero))))).
```

```
?- somme(X, Y, s(s(zero))).
```

```
    X = zero    Y = s(s(zero)) ;
```

```
    X = s(zero)  Y = s(zero) ;
```

```
    X = s(s(zero))  Y = zero
```

Donner les bonnes propriétés du prédicat `somme/3` et des exemples illustratifs.

- f. Définir de la même façon la multiplication et donner des utilisations intéressantes.

- g. Ecrire le prédicat `pair/1` qui ne réussit que si l'entier spécifié est pair.

- h. Ecrire le prédicat `inf/2` qui ne réussit que si le premier paramètre est inférieur au second.

## 2. Arithmétique

- a. Ecrire le prédicat `puissance(X, N, V)` qui ne s'efface que si  $V=X^N$ .

- b. Ecrire le prédicat `entre(I, J, K)` qui ne s'efface que si  $K$  est un entier entre  $I$  et  $J$  (par `backtrack`, tous ces nombres doivent être générés).

## 3. Prédicats exotiques

Ecrire les prédicats suivants :

- a. `permutation(L1, L2)` qui réussit si  $L2$  est une permutation de la liste  $L1$ .

- b. `sousSomme(L1, N, L2)` qui réussit si  $L1$  est une liste d'entiers, et  $L2$  est une liste d'éléments de  $L1$  dont la somme vaut  $N$ .

- c. `evenLength(L)` qui réussit si  $L$  est de longueur paire. Ecrire de manière similaire le prédicat `oddlength/1`.

- d. `add(L, N)` qui réussit si  $N$  est égal à la somme des éléments de la liste  $L$ . Donner aussi une version inverse ( $N$  en « entrée »).

- e. `double(L1, L2)` qui réussit si la liste  $L2$  est la liste  $L1$  dont les éléments ont été dupliqués.

- f. `last(X, L)` qui réussit si  $X$  est le dernier élément de la liste  $L$ . Donner deux versions différentes.

- g. `sousListe(L1, L2)` qui réussit si  $L1$  est une sous-liste de  $L2$ .