



INSA Lyon
20, avenue Albert Einstein
69621 Villeurbanne Cedex

LIVRABLE DE PROJET

Prolog

« Puissance 4 »

du 1^{er} au 15 octobre 2013



Hexanôme H4404 :

Guillaume ABADIE
Nicolas BUISSON
Louise CRÉPET
Rémi DOMINGUES
Aline MARTIN
Martin WETTERWALD

Enseignants :

Jean François BOULICAUT
Mehdi KAYTOUE

Année scolaire 2013-2014

1. Bilan des exercices

1.1 Predicats

La particularité Prolog reside dans le fait qu'il n'y a plus d'itérations comme un langage de programmation conventionnel. Tout n'est que predicat. Ainsi, la méthode de programmation change de façon de penser. Fini le traitement de données dans des variables que l'on met jour, car la programmation predicat permet simplement de lier des propriétés entre des variables et/ou constantes.

Considérons pour la suite, le predicat :

$$(membre(X, L) \Leftrightarrow vrai) \Leftrightarrow X \in L$$

1.2 Vérification de propriétés

La vérification de propriétés permet s'assurer que une (ou plusieurs) constantes vérifient un ensemble de predicats. Considérons le code ci-dessous :

Alors on a à l'exécution :

```
?- membre(1, [1, 2, 3]).  
true
```

```
?- membre(4, [1, 2, 3]).  
false
```

En effet, à la première interrogation, on vérifie le prédicat $1 \in [1, 2, 3]$ ce qui est vrai, d'où la réponse de Prolog 'vrai'. La propriété entre ces deux paramètres est alors vérifiée, renvoyant ainsi vrai. Tandis que la seconde interrogation $4 \in [1, 2, 3]$ est fausse car $4 \notin [1, 2, 3]$, d'où la réponse 'false'.

1.3 Reprouvabilité

La reprovabilité consiste maintenant de définir des propriétés entre des variables et/ou constantes. Par exemple :

$$X \in [1, 2, 3].$$

Ce qui en Prolog donne :

```
?- membre(X, [1, 2, 3]).
```

On lis a ce moment que X compose la liste constante $[1, 2, 3]$. Ainsi a l'exécution, Prolog peut evaluer les solutions de L grace a cette propriete ainsi définie :

```
?- membre(L, [1, 2, 3]).
L = 1;
L = 2;
L = 3;
false
```

1.4 Reprouvabilité non-déterministe

La dangeureusitee de de la reprouvabilité, est qu'il est possible qu'une infinitée de solutions vérifient une meme propriété. Considerons par exemple le code suivant :

```
?- membre(1, L).
```

Cette est equivalent à $1 \in L$. Mais alors, combien de listes pourraient vérifié cette propriété?

Initialisation : Une liste telle que $[1, 2]$ vérifie cette propriété.

Hérédité : En notant $cat(A, B)$ la concatenation de deux listes A et B ,

Soit une liste L telle que $1 \in L$,

Alors $\forall X \in \mathbb{N} / 1 \in cat([X], L)$

Conclusion : Il existe une infinité de solutions et Prolog va essayer de toutes les générer, causant une exception du au manque de mémoire de la machine.

```
?- membre(1, L).
L = [1|_G2214] ;
L = [_G2213, 1|_G2217] ;
L = [_G2213, _G2216, 1|_G2220] ;
L = [_G2213, _G2216, _G2219, 1|_G2223] ;
L = [_G2213, _G2216, _G2219, _G2222, 1|_G2226] ;
L = [_G2213, _G2216, _G2219, _G2222, _G2225, 1|_G2229] ;
...
```

1.5 Programation de prédicats

Au paravant, nous fesions que utiliser des prédicats, mais bien entendu, l'objectif est de pouvoir coder les siens. Pour cela, interessons nous a la reecriture de

$$(membre(X, L) \Leftrightarrow vrai) \Leftrightarrow X \in L$$

```
membre(X, [X|_]) .  
membre(X, [_|L]) :- membre(X, L) .
```

2. Projet : Puissance 4