

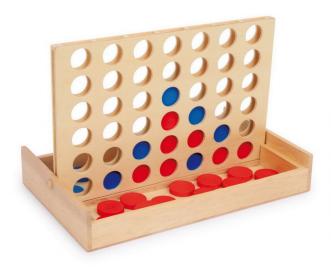
INSA Lyon 20, avenue Albert Einstein 69621 Villeurbanne Cedex

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 éxercices

1.1 Predicats

La particularite Prolog reside dans le fait qu'il n'y a plus d'iterations comme un language de programmation conventionnel. Tout n'est que predicat. Ainsi, la methode de programmation change de facon de penser. Fini le traitement de donnes dans des variable que l'on met jour, car la programmation predicat permet simplement de lier des proprietes entre des variables et/ou constantes.

Considerons 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 (out plusieur) constantes verifies un ensemble de predicat. Considerons le code ci dessous :

Alors on a à l'éxécution:

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

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

En effet, a la premiere intérogation, on vérifie le prédicat $1 \in [1, 2, 3]$ ce qui est vrai, d'ou la reponse de Prolog 'vrai'. La proprieté etre ces deux paramêtres est alors vérifié, renvoyant ainsi vrai. Tandis que la seconde interogation $4 \in [1, 2, 3]$ est fausse car $4 \notin [1, 2, 3]$, d'ou la reponse 'false'.

1.3 Reprouvabilitée

La reprouvabilitées consiste maintenant de definir des propriete 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'execution, Prolog peut evaluer les solutions de L grace a cette proprietee ainsi définie :

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

1.4 Reprouvabilitée multiple

Une propriétée sur une variable par exemple, peut etre definit par plusieur predicats. Par exemple :

$$\begin{cases}
L \in [1, 2, 3] \\
L \in [3, 4, 2]
\end{cases}$$

Cela revient simplement a l'ecriture en Prolog :

```
?— membre(L, [1, 2, 3]), membre(L, [3, 4, 2]). 
 L = 2; 
 L = 3; 
 false
```

1.5 Reprouvabilitée non-déterministe

La dangeureusitee de la reprouvabilitée, est qu'il est possible qu'une infinitée de solutions vérifient une meme propriétée. 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 éxiste 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];
```

1.6 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 reccriture de

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

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

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

2. Projet : Puissance 4