

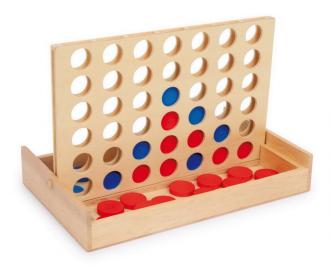
INSA Lyon 20, avenue Albert Einstein 69621 Villeurbanne Cedex

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 Prédicats

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

Considérons pour la suite, le prédicat :

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

1.2 Vérification de propriétés

La vérification de propriétés permet de s'assurer qu'une (ou plusieurs) constantes vérifient un ensemble de prédicats. Considérons le code ci-dessus.

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 « true ». La propriété est alors vérifiée, renvoyant ainsi vrai. Tandis que la seconde interogation $4 \in [1, 2, 3]$ est fausse car $4 \notin [1, 2, 3]$, d'où la réponse « false ».

1.3 Reprouvabilité

La reprouvabilité consiste à 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'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 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). 
  \begin{array}{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] \ ; \\ \dots \end{array}
```

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 reccriture de

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

3

```
\begin{array}{ll} \operatorname{membre}(X, & [X|\_]\,)\,. \\ \operatorname{membre}(X, & [\_|L]\,) \,:- \,\operatorname{membre}(X, \;L\,)\,. \end{array}
```

2. Projet : Puissance 4