



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 :

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

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 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 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'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 multiple

Une propriété sur une variable par exemple, peut etre definit par plusieurs predicats. Par exemple :

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

Cela revient simplement a l'écriture en Prolog :

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

1.5 Reprouvabilité non-déterministe

La dangerosité de la reprovabilité, est qu'il est possible qu'une infinité 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 vérifier 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] ;
...

```

1.6 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 reecriture de

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

```

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

```

2. Projet : Puissance 4

2.1 Règles du jeu

Le puissance 4 est un jeu de société à deux joueurs. Chaque joueur doit, à son tour, insérer un jeton de sa couleur dans une des sept colonnes côte à côte, chacune ayant une capacité maximale de six jetons. Le but du jeu est d'aligner verticalement, horizontalement ou en diagonale, 4 jetons de sa couleur avant l'adversaire.

2.2 But du joueur idéal

Dans le cas d'un joueur idéal, le but n'est simplement d'aligner 3 jetons précédemment, puis prévoir de jouer le 4^{ème} au tour suivant. En effet, l'adversaire pourrait bloquer cet alignement, lorsque c'est à son tour de jouer. L'objectif du joueur idéal est donc de réaliser au moins deux alignements de 3 jetons en un coup. Laissant ainsi le joueur adverse contre l'inévitable fatalité : Il ne peut plus contrer ces alignements en un seul jeton.

2.3 Travail réalisé

En plus de l'implémentation du module de mécanisme de jeu et réalisation des tests unitaires, nous avons implémenté 4 joueurs ayant des stratégies différentes :

- joueur aléatoire ;
- joueur aléatoire muni d'heuristiques ;
- joueur parcourant l'arbre des possibilités ;
- intelligence artificielle apprenant par **moteur d'inférence**, de ses échecs précédents.

Mais aussi :

- interface utilisateur en ligne de commande pour jouer une partie ;
- module de tournoi générant des statistiques ;
- module d'entraînement du moteur d'inférence ;
- module d'étude de l'apprentissage du moteur d'inférence ;
- sauvegarde et chargement de la base de connaissances du moteur d'inférence.