TP6 – Système à base de règles en Prolog

Arthur Aubret, Hugo Castaneda, Rémy Chaput,
Nathalie Guin, Marie Lefevre

SYSTEMES A BASE DE REGLES

On considère des problèmes que l'on peut résoudre à l'aide d'un système à base de règles. Nous allons, dans un premier temps, définir un moteur d'inférences à chaînage avant que nous appliquerons sur un problème jouet. Vous définirez ensuite la base de règles relative à un problème de fleurs. Attention, votre moteur doit être générique et fonctionner sur le problème jouet, le problème de fleurs mais également sur tout autre jeu de test (une autre base de règles).

PARTIE 1 : MOTEUR EN CHAÎNAGE AVANT

Notre problème jouet est décrit comme suit. Nous disposons de 6 règles :

- R1 : A et B → C
- R2 : C et non(D) → F
- R3 : F et B → E
- R4 : F et A → non(G)
- R5 : non(G) et F → B
- R6 : A et H → L

Nous connaissons trois faits : A, C et non(D), et nous voulons démontrer le fait E.

Nous allons réaliser un moteur d'inférences en chaînage avant qui fonctionne comme sur l'exemple suivant :

```
?- raz.                    /* on vide la base de faits */
true.

?- faits([a,c,non(d)]).    /* on charge les nouveaux faits */
true.

?- saturer.                /* on lance le moteur en chainage avant */
r2 : non(d) a c f
r4 : non(d) non(g) a c f
r5 : non(d) non(g) a c f b
r1 : non(d) non(g) a c f b
r3 : non(d) non(g) a c f b e
true.
```

Pour réaliser ce moteur d'inférences en chaînage avant, il faut :

Question 1 : Définir la base de règles sous forme de listes composées de listes de prémisses et de listes de conclusions, tel que :

```
regle(Ri, ListeDePrémisses, ListeDeConclusions).
```

Question 2 : Définir un prédicat permettant à l'utilisateur d'initialiser la base de faits. On utilisera le prédicat `assert` pour ajouter `vrai(Fait)` pour les faits positifs et `faux(Fait)` pour les faits négatifs.

Question 3 : Définir un prédicat permettant à l'utilisateur de vider la base de faits. On utilisera le prédicat `retractall` pour supprimer les faits.

Question 4 : Définir un prédicat `saturer` qui sature la base de règles et produit une trace de son fonctionnement. L'algorithme utilisé pour ce moteur sera le suivant :

```
Changement <-- Vrai
Tant que Changement est Vrai
      Changement <-- Faux
      Boucle sur les règles : soit R une règle de BaseRègles
            Si R n'est pas marquée
            et si les prémisses de R appartiennent à BaseFaits
            Alors ajouter les conclusions de R à BaseFaits
                  changement <-- Vrai
                  marquer R
            FinSi
      FinBoucle
FinTantQue
```

Rappel 1 : La seule boucle qui existe en Prolog est la boucle mue par l'échec.

Rappel 2 : Lorsque vous voulez utiliser un prédicat relatant un fait (par exemple `vrai/1`) dans un autre prédicat, mais que votre base de faits n'est pas encore remplie, vous devez le déclarer dynamiquement :

```
:- dynamic vrai/1, faux/1.
```

PARTIE 2 : NOUVELLE BASE DE CONNAISSANCES

Question 5 : Tester votre moteur d'inférences sur la base de règles florales ci-dessous :

r2 :	Si phanerogame et graine_nue	Alors sapin et ombre
r1 :	Si fleur et graine	Alors phanerogame
r3 :	Si phanerogame et un_cotyledone	Alors monocotyledone
r4 :	Si phanerogame et deux_cotyledone	Alors dicotyledone
r5 :	Si monocotyledone et rhizome	Alors muguet
r6 :	Si dicotyledone	Alors anemone
r15 :	Si joli	Alors non rhizome
r7 :	Si monocotyledone et non rhizome	Alors lilas
r8 :	Si feuille et non fleur	Alors cryptogame
r9 :	Si cryptogame et non racine	Alors mousse
r10 :	Si cryptogame et racine	Alors fougere
r11 :	Si non feuille et plante	Alors thallophyte
r12 :	Si thallophyte et chlorophylle	Alors algue
r13 :	Si thallophyte et non chlorophylle	Alors champignon et non comestible
r14 :	Si non feuille et non fleur et non plante	Alors colibacille

Question 6 : Qu'obtenez-vous avec les deux bases de faits suivantes :

- fleur, graine et dicotylédone
- fleur, graine.

BONUS : MOTEUR EN CHAINAGE ARRIERE

Question 7 : Écrire un moteur d'inférences d'ordre 0 fonctionnant en chaînage arrière. On représentera la base de règles et la base de faits comme dans la première partie.

Pour rappel, le chainage arrière consiste à assigner un but au système et à unifier la partie conclusion des règles avec ce but. Les nouveaux buts à démontrer sont alors les prémisses de la règle sélectionnée. Si les prémisses ne sont pas dans la base de faits, ni démontrables *via* une autre règle, vous ferez échouer la démonstration.

Sur notre problème jouet, son exécution pourra donner par exemple :

```
?- raz.                    /* on vide la base de faits */
true.

?- faits([a,c,non(d)]).    /* on charge les nouveaux faits */
true.

?- satisfait(e) .          /* on cherche à démontrer E */
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
a dans la base de faits
non(g) satisfait grace a r4
c dans la base de faits
non(d) dans la base de faits
f satisfait grace a r2
b satisfait grace a r5
e satisfait grace a r3
true

?- satisfait(l).         /* on cherche à démontrer L */
a dans la base de faits
false.
```

Question 8 : Qu'obtenez-vous avec la base de règles des fleurs, la première base de faits de la question 6 et le but « Muguet » ? et avec le but « Lilas » ?

Question 9 : Pourquoi ? Comment remédier au problème ?