Université de Technologie de Compiègne

Durée : Deux Heures

Les documents ne sont pas autorisés
Tous ordinateurs, toutes communications sont interdits

Utilisez trois copies séparées :

- une copie pour le problème n°1
- une autre copie pour le problème n°2
- une autre copie pour le problème n°3

I Questions de cours (5 points)

1 - Donnez trois approches de l'IA. Pour chacune d'elle, précisez ses particularités.

2 - Qui a introduit la notion de frame ? En quoi consiste cette notion ?

3 - En quoi consistent les logiques de description ? Quel en est le premier représentant ?

4 - Quel est l'intérêt des méthodes de programmation des algorithmes génétiques par rapport aux méthodes traditionnelles de résolution de problème ? Précisez la classe de problèmes pour laquelle les méthodes d'algorithmes génétiques sont particulièrement adaptées.

5 - Que signifie le sigle RNA ? Donnez la définition de la notion signifiée.

II Des champignons... (7.5 points)

Pour essayer d'aider les mycologues distingués à analyser leur cueillette, on se propose de construire une base de données contenant la description des différentes espèces, puis un petit système expert pour aider à reconnaître si un échantillon est comestible.

On prendra comme exemple trois champignons à lamelles :

Amanite : champignon dont le chapeau a une couleur vert-jaunâtre ou blanchâtre, dont la chair est blanche et a une odeur faible, dont les lamelles sont blanches et de forme inégale. Ce champignon est mortel.

Coprin : champignon dont le chapeau est blanc et ovoïde, le pied est cylindrique, blanc et creux, la chair est légère et aqueuse, de saveur agréable. Les lamelles sont denses et serrées, minces et hautes. Ce champignon est comestible, délicat lorsqu'il est jeune.

Chanterelle : ce champignon a un chapeau en forme d'entonnoir, de couleur jaune d'œuf, un

pied plein, une chair de couleur jaune et d'odeur agréable. Il est comestible et excellent.

On va construire pour chaque champignon une description de celui-ci sous forme d'une liste que l'on associera à son nom. C'est-à-dire que la valeur du symbole COPRIN sera la liste décrivant ce champignon. La base de données sera la liste :

```
*CHAMPIGNONS* = (AMANITE COPRIN CHANTERELLE)
```

Chaque liste décrivant un champignon aura la forme suivante :

```
<description> ::= (<description d'un élément>*)
<description d'un élément> ::= (<élément> (<desc. d'une caractéristique>*))
<desc. d'une caractéristique> ::= (<propriété> . <liste de valeurs possibles>)
<liste de valeurs possibles> ::= (<valeur>*)
```

où * indique une répétition possible (un ou plusieurs termes).

2.1 Donner la liste des éléments (0.5 pt). Note : on considérera la comestibilité comme un élément.

<élément> ::= CHAPEAU | ...

2.2 Donner la liste des propriétés pour tous les éléments (0.5 pt). Attention! Il se peut que quelques informations soient manquantes. À vous d'imaginer une solution adéquate.

<propriété> ::= COULEUR | ...

2.3 Donner les descriptions des 3 champignons (1 pt).

On suppose que les descriptions ont été introduites dans notre base. Il est donc nécessaire de disposer d'une fonction d'interrogation que nous appellerons `inter`, utilisable comme suit :

```
? (inter 'couleur 'chapeau 'amanite)
(VERT-JAUNATRE BLANCHATRE)
```

2.4 Définir la fonction `inter` (2 pts).

Pour reconnaître un champignon de la cueillette, on va utiliser un mini système expert.

2.5 Donner un schéma succinct de la structure d'un système expert (1 pt).

Dans notre système expert, chaque règle aura le format suivant :

```
<règle> ::= (<nom de la règle> SI <prémisse>* ALORS <conclusion>*)
<prémisse> ::= (<élément> <propriété> <liste de valeurs possibles>)
<conclusion> ::= (<élément> <propriété> <liste de valeurs possibles>)
<liste de valeurs possibles> ::= (<valeur>*)
```

où * représente un répétition possible des termes (un ou plusieurs termes).

Pour passer de la description d'un champignon à une règle du système expert, il est nécessaire de transformer une description de la base de données en une règle. Il faudra en particulier fabriquer les prémisses de la règle.

On utilisera pour cela plusieurs fonctions progressivement plus complexes.

La fonction `faire-clause` traite un élément n'ayant qu'une caractéristique :

```
? (faire-clause '(chapeau (couleur vert-jaunatre blanchatre)))
(CHAPEAU COULEUR (VERT-JAUNATRE BLANCHATRE))
```

2.6 Écrire cette fonction (0.5 pt).

La fonction `faire-clauses` traite un élément pouvant avoir plusieurs caractéristiques. Le résultat peut donc être une liste de plusieurs prémisses (**2 pts)** :

```
? (faire-clauses '(chapeau (couleur vert-jaunatre blanchatre)
                           (forme bombee)))
((CHAPEAU COULEUR (VERT-JAUNATRE BLANCHATRE))(CHAPEAU FORME (BOMBEE)))
```

2.7 Écrire cette fonction.

III L'inférence (7.5 points)

Plus généralement il faut passer de la description du champignon à la liste des prémisses. Ce sera le rôle de la fonction `faire-liste-premisses`.

3.1 Écrire cette fonction (1.5 pts).

```
? (defun faire-liste-premisses (nom-champignon)
« Prend un nom de champignon (ex : coprin) et retourne la liste de
prémisse correspondant à la description. »
```

Par ailleurs on dispose du code suivant :

```
? (defvar *liste-de-regles* nil)
*LISTE-DE-REGLES*

? (defun ff (set)
    (dolist (item set)
      (let ((new (gentemp "R-")))
        (set new `(,new SI ,@(faire-liste-premisses item)
                        ALORS ,@(faire-clauses
                                  (assoc 'comestibilite (symbol-value item)))))
        (push new *liste-de-regles*)))
    (reverse  *liste-de-regles*))
```

3.2 Que fait la fonction ff ? (2 pts)

Le moteur d'inférence utilisé pour notre système expert est décrit par la fonction `inference` :

```
(defun inference (BR R RDE *faits*)
  "BR: base de règles, R: règle courante, RDE: règles déjà examinées."
  (cond ((cdr (assoc 'comestibilite *faits*)))
        ((null BR) nil)
        ((null R)(inference BR (car BR) RDA *faits*))
        ((applicable? R *faits*)
         (inference (remove R BR) nil RDE (append (conclusions R) *faits*)))
        ((inference (remove R BR)(cadr BR)(cons R RDE) *faits*))))
```

La base de faits a la forme d'une liste de triplets :

```
  ? *faits*
  ((CHAPEAU COULEUR JAUNE)(CHAIR GOUT AGREABLE)...)
```

3.3 Donner succinctement en français l'algorithme correspondant à la fonction `inference` (2 pts).

Noter la fonction `applicable?` utilisée dans la fonction `inference` qui teste si une règle

s'applique pour une base de faits donnée.

On supposera que pour chaque règle l'on dispose des fonctions `premisses` et `conclusions` permettant d'obtenir la liste de prémisses et des conclusions de la règle à partir de son nom, et que l'on a une fonction `match` qui permet de savoir si une prémisse est dans la base de faits :

```
? (premisses 'R-5)
((CHAPEAU COULEUR (BLEU BLANC ROUGE))(CHAPEAU FORME (POINTUE))...)

? (defun match (premisse *faits*)
    "teste si les faits contiennent un triplet qui valide la prémisse."
    (dolist (fait *faits*)
      (when (and (eq (car fait) (car premisse))
                 (eq (cadr fait)(cadr premisse)))
        (return-from match (member (caddr fait) (caddr premisse))))))
```

3.4 Écrire la fonction `applicable?` (2 pts)

A partir de ce qui précède, on obtient :

```
? (setq *faits* ; description de l'échantillon
    '((lamelles presence t)(chair couleur jaune-d-oeuf)
      (chair odeur agreable)(chapeau couleur jaune)
      (chapeau forme entonnoir))
...
? (ff *champignons*) ; constitution de la liste de règles
...
? (inference *liste-de-regles* nil nil *faits*) ; analyse
(VALEUR (COMESTIBLE EXCELLENT)) ; résultat
```