IA01 – Automne 2006
Examen Final

Durée : Deux Heures
Les documents ne sont pas autorisés

Utilisez trois copies distinctes :
- une copie pour la partie I,
- une copie pour la partie II,
- une dernière copie pour la partie III.

I [5 points] - Qu'avez-vous retenu du cours ?

1 - Qui a introduit le modèle du neurone formel ? En quoi consiste ce modèle ?
2 - En quoi consiste une recherche dans un espace d'états ? Donner la définition formelle d'un problème de recherche.
3 - Quel est l'intérêt des méthodes de programmation des algorithmes génétiques par rapport aux méthodes traditionnelles de résolution de problème ?
4 - A quoi sert la représentation des connaissances ? Donnez deux formalismes de représentation.
5 - Qu'est-ce que le RàPC ? Donner les trois grandes familles de modèles de RàPC.

II [8 points] - Matching

On a vu en TD un exemple de fonction matching permettant d'utiliser un joker qui apparie 0 ou 1 mot :

```
(defun match-31 (pattern text)
   (cond
   ((null pattern) t)
   ((null text)(if (eq (car pattern) '*)
    (match-31 (cdr pattern) text) nil))
   ((and (equal (car pattern)(car text))
         (match-31-rest (cdr pattern)(cdr text))))
   ((and (eq (car pattern) '*)
         (or (match-31-rest (cdr pattern) text)
             (match-31-rest (cdr pattern)(cdr text)))))
   (t(match-31 pattern (cdr text)))))

(defun match-31-rest (pattern text)
  (cond
   ((null pattern) t)
   ((null text) (if (eq (car pattern) '*)
    (match-31-rest (cdr pattern) text) nil))
   ((equal (car pattern)(car text))
    (match-31-rest (cdr pattern)(cdr text)))
   ((eq (car pattern) '*)
    (or (match-31-rest (cdr pattern) text)
        (match-31-rest (cdr pattern)(cdr text))))))
```

On se propose de modifier *match-31* afin que le joker (*) apparie 0, 1 ou plusieurs mots. La fonction à programmer se nomme *match-final*. Notons, qu'il ne sera plus permis d'avoir plusieurs jokers consécutifs.

1 – Donner un exemple d'appel de *match-final* permettant de reconnaître dans un texte la présence d'un *a* suivi à une distance quelconque d'un *b*.

2 – La fonction *position* en lisp permet de retourner la position d'un élément dans une liste ou bien *nil* s'il n'est pas présent.

> (position 'a '(a b c d))
0
> (position 'v '(a b c d))
NIL

Programmer une votre propre fonction *position* : *my-position*

3 – La fonction *subseq* en lisp permet de retourner une sous-liste d'une liste à partir de l'indication de la position d'éléments de la liste.

> (subseq '(a b c d) 0 2); retourne la sous-liste constituée des éléments placés de la 0 à la
(a b) ; 2ème place
> (position '(a b c d) 2) ; retourne la sous-liste à partir de l'élément positionné à la deuxième
(c d) ; place

Programmer en lisp votre propre fonction *subseq* : *my-subseq*
Indice : penser à utiliser les arguments optionnels (&optional)

4 – A l'aide des fonctions *my-positio*n et *my-subseq*, donner un algorithme de la fonction *match-final*.

5 – Donner le code lisp de l'algorithme proposé pour la fonction *match-final*.

III [7 points] - **Représentation Objet**

Dans cet exercice, nous nous proposons de créer un modèle de représentation objet des UV à l'UTC. Ce modèle doit rendre compte des concepts associés à une UV, à un étudiant et à un enseignant. Une UV est caractérisée par son nom, des intervenants (responsable, intervenants cours, chargés TD), des horaires de cours, de TD et de TP (différents groupes), etc. Un étudiant est caractérisé par son identité, les UV qu'il suit, ses notes, etc. Un enseignant est caractérisé par son identité, les UV qu'il dispense (cours, TD, TP), etc.
La représentation choisie permettra, par la suite, de trouver la liste des étudiants qui suivent l'UV IA01 et quels sont leurs résultats.

NB : on utilisera une seule liste pour tous les intervenants, une seule pour tous les horaires et une seule liste pour toutes les notes (on pourra utiliser une a-list).
Notes : ((IA01 . 15) (NF28 . 12) ...)

1 – Donner une représentation objet de *$UV*, $*Etudiant* et $*Enseignant*

2 – Définir en CLOS les classes précédentes

Rappel

(DEFCLASS <nom-classe> (<nom-de-superclasse>*)
(<description-slot>*)
(<option-classe>*))

La description d'un slot (ou champ) est de la forme (<nom-slot> <option-slot>*), où chaque option est un mot clé suivi d'un nom, d'une expression ou d'une autre forme.

Les options peuvent être :

:ACCESSOR < fonction>: nom de la fonction pour l'accès en lecture/écriture
:INITFORM <expression> : valeur initiale du champ par défaut
:INITARG <symbole> : symbole utilisé comme mot-clé dans la liste des arguments lors de la création d'une instance
:DOCUMENTATION <chaîne de caractères> : description sémantique du champ
:TYPE <spécification-type> : indique que le contenu du champ est toujours du type spécifié
…

3 – Créer une instance de *$UV ($IA01*)

Rappel
Pour créer des instances d'une classe il faut utiliser la fonction MAKE-INSTANCE:

(MAKE-INSTANCE <classe> {valeurs d'initialisation des champs}*)

4 – Définir une méthode de la classe *$UV* qui permette d'afficher le nom de tous les étudiants inscrits ainsi que la note qu'ils ont obtenue dans cette UV.

Rappel
Une méthode est définie par:

(DEFMETHOD <nom-fonction-générique> <liste-spécialisée> <forme>*)

Chaque paramètre apparaissant dans <liste-spécialisée> peut être spécialisé, c'est-à-dire qu'il peut être défini comme étant d'un certain type ou d'une certaine classe. La <liste-spécialisée> peut être de la forme:

((var1 type1) (var2 type2) ... varN ...)