

Examen médian IA01 A09
Université de Technologie de Compiègne

Durée : 2H

Les documents ne sont pas autorisés

Tous ordinateurs, toutes communications sont interdits

Utilisez quatre copies séparées :

- *une copie pour la partie I*
- *une autre copie pour la partie II*
- *une autre copie pour la partie III*
- *une autre copie pour la partie IV*

1. Qu'avez-vous retenu du cours (4 points) ?

- 1) Donnez trois approches de l'IA en précisant leurs particularités.
- 2) A quoi sert la représentation des connaissances ? A quel type d'approche se rattache-t-elle ?
- 3) Donnez un exemple de formalisme de représentation des connaissances. Explicitez-le.
- 4) Donnez et expliquez les principales caractéristiques du langage Lisp.

2. Système expert (6 points)

On considère un système expert permettant de déterminer si une ville donnée mérite le voyage, avec les règles suivantes :

- **R1 : Si** belle ville **et** très bon restaurants **alors** ville méritant le voyage.
- **R2 : Si** ville historique **alors** ville méritant le voyage.
- **R3 : Si** autochtones accueillants **et** traditions folkloriques **alors** ville méritant le voyage.
- **R4 : Si** monuments **et** végétation abondante **alors** belle ville.
- **R5 : Si** tradition culinaire **alors** bons restaurants.
- **R6 : Si** restaurants 3 étoiles **alors** très bons restaurants.
- **R7 : Si** restaurants 3 toques **alors** très bons restaurants.
- **R8 : Si** musées **et** ville ancienne **alors** ville historique.
- **R9 : Si** Provence **et** bord de mer **alors** autochtones accueillants.
- **R10 : Si** parcs verdoyants **et** avenues larges **alors** végétation abondante.

1) Précisez l'ordre de ce système expert. Vous appellerez également la signification de cette notion d'ordre, et vous illustrerez vos propos par des exemples simples pour chacun des cas.

2) On considère une ville qui possède les caractéristiques suivantes :

parcs verdoyants, avenues larges, monuments, restaurants 3 toques, ville ancienne

Le système expert permet-il de prouver que cette ville mérite le voyage ?

Simulez le fonctionnement du moteur en chaînage avant et en profondeur d'abord. Pour chaque cycle du moteur, vous préciserez l'ensemble des conflits, la règle choisie et l'état de la base de faits.

3) Dessinez l'arbre "et-ou" associé à un fonctionnement du moteur en chaînage arrière et en profondeur d'abord.

Proposez une représentation pour une règle, pour un fait et pour la base de règles et la base de faits.

4) Ecrivez une fonction `regles-candidates` (But Regles) renvoyant la liste des règles candidates pour prouver le but But parmi les règles de la liste Regles.

5) Proposez un algorithme (clairement présenté et écrit en français) effectuant le parcours de l'arbre que vous avez tracé à la question 3. Vous utiliserez pour cela la fonction `regles-candidates` et les fonctions de service suivantes :

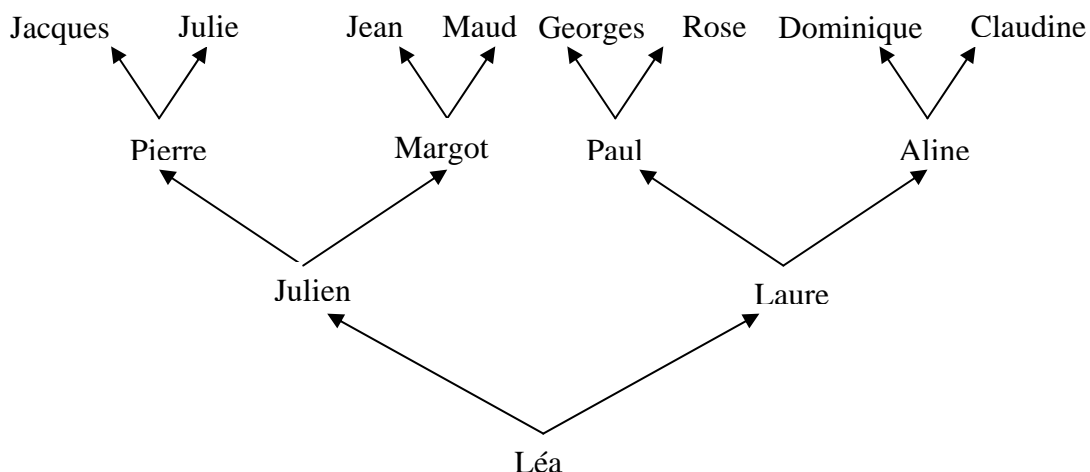
- `premisses (R)` : permet d'obtenir les prémisses d'une règle R
- `conclusion (R)` : permet d'obtenir la conclusion de la règle R
- `vrai? (fait)` : permet de savoir si fait est dans la base de faits (avec la valeur vrai).

N. B. : il n'est pas nécessaire de donner les algorithmes (ni les fonctions Lisp) correspondant à ces fonctions de service

3. Programmation Lisp (4 points)

On considère un arbre généalogique.

Exemple :



Une personne possède un père et une mère. On choisit de représenter les arbres de ce type de la façon suivante : chaque personne est représentée par son nom suivi de celui de son père puis celui de sa mère.

Ainsi, avec l'exemple, on obtient :

((Léa Julien Laure) (Julien Pierre Margot) (Laure Paul Aline) (Pierre Jacques Julie) (Margot Jean Maud) (Paul Georges Rose) (Aline Dominique Claudine))

1) Ecrivez deux fonctions *Pere* et *Mere* qui renvoient respectivement le nom du père ou de la mère d'une personne passée en paramètre.

Exemples :

> (Pere 'Léa)	> (Mere 'Léa)
> Julien	> Laure
> (Pere 'Jeannette)	> (Mere 'Jeannette)
> nil	> nil

2) Ecrivez une fonction *GrandsPeres* qui renvoie une liste composée des deux grands pères.

Exemples :

> (GrandsPeres 'Léa)	> (GrandPere 'Jean)
> (Pierre Paul)	> nil

3) Généralisez cette fonction afin d'obtenir une fonction *Ancetre* qui renvoie une liste comprenant tous les ancêtres masculins d'une personne (ne considérez que les liens paternels).

Exemples :

> (Ancetre 'Léa)	> (Ancetre 'Jean)
> (Julien Pierre Jacques)	> nil

4) Ecrivez une fonction *Enfant* qui renvoie l'enfant d'une personne.

Exemples :

> (Enfant 'Léa)	> (Enfant 'Aline)
> nil	> Laure

5) Ecrivez une fonction *Descendant?* qui renvoie vrai si le premier paramètre est un descendant du second.


Exemples :

> (Descendant? 'Léa 'Aline)
> t
> (Descendant? 'Aline 'Rose)
> nil

4. Parcours de chemin d'un Personnage Non Joueur (PNJ) (6 points)

Vous devez programmer un jeu vidéo. Vous devez proposer des fonctions qui permettront à des personnages non-joueurs (PNJ) de se déplacer dans un environnement inconnu. Vous devez doter ces personnages de capacités de perception, de décision et d'action. L'arbre des chemins possibles n'est pas construit à l'avance. Au fur et à mesure que le PNJ avance, il perçoit l'environnement. Vous simulerez cette perception en demandant à l'utilisateur quels sont les états que le PNJ peut accéder. L'environnement dans lequel évolue votre PNJ est constitué d'une entrée, d'une sortie, de pièces à traverser et d'obstacles franchissables ou non selon les capacités de votre PNJ (e.g. il peut s'agir de murets qu'un PNJ doté de capacités sportives peut franchir et qu'un PNJ doté de capacités normales ne peut franchir). Un PNJ peut se déplacer en horizontal, en vertical mais pas en diagonal. On peut imaginer un environnement comme celui-ci :

E			27	26	
1	2	3	24	25	S
6	5	4	23	22	
7	8	9	20	21	
12	11	10	19	18	
13	14	15	16	17	

 Obstacle franchissable

Vous disposez d'un formalisme de représentation de cet environnement qui permettra de mettre à jour et d'enrichir la représentation du PNJ au fur et à mesure de l'exploration. On propose un formalisme de représentation sous forme de a-list qui permette de préciser les obstacles et les espaces libres. Voici un début de représentation de l'environnement d'exemple que le PNJ pourrait avoir après avoir exploré les états E, 1, 2 et 3 :

```
> *graph*
> ((E (1 . libre)) (1 (E . libre) (2 . libre) (6 . libre)) (2 (1 . libre)
(3 . libre) (5 . obstacle)) (3 (2 . libre) (4 . obstacle)))
```

1) Donnez la définition formelle de ce problème de recherche dans un espace d'état (ensemble des états accessibles, ensemble des états initiaux, ensemble des états solutions, ensembles des actions, pour chaque état ensemble des successeurs possibles par action) (1 point)

2) Proposez une fonction de service « perception » qui construit les successeurs de l'état courant en demandant à l'utilisateur de rentrer les états suivants selon le formalisme proposé. Cette perception est alors indépendante des capacités du PNJ. La fonction « perception » ajoutera cette liste à la représentation du PNJ.

Rappel : vous utiliserez la fonction (read) pour saisir la perception.

Exemple :

```
> (perception 6 *graph*)
Vous êtes dans l'état 6, entrez les états possibles perçus par le PNJ (sous
forme de a-list) : (6 (1 . libre) (5 . obstacle) (7 . libre))
```

Le résultat dans *graph* pourra alors être comme suit :

```
((E (1 . libre)) (1 (E. libre) (2 . libre) (6 . libre)) (2 (1 . libre) (3.
libre) (4 . obstacle)) (3 (2 . libre) (4 . obstacle)) ((6 (1 . libre) (5 .
obstacle) (7 . libre)))
```

3) Proposez une fonction de service qui cherche les successeurs valides à un état courant donné. Les successeurs valides étant les successeurs qui n'ont pas encore été explorés et qui peuvent être franchis par le PNJ. Il faudra pour cela prendre en compte les capacités du PNJ selon le formalisme donné (0 : capacités normales, 1 : capacités sportives).

Exemple :

Le PNJ1 a des capacités sportives, il est dans l'état 2. L'appel de la fonction (successeurs-valides etat *graph* anciensEtats capacitesPNJ) se ferait ainsi :

```
> (successeurs-valides 2 *graph* `(E 1 2) 1)
```

Pour l'état 2 si le PNJ vient de 1 et qu'il est sportif, la fonction renvoie : ((3 . libre) (5 . obstacle))

4) Donnez les grandes lignes de l'algorithme qui permettrait au PNJ d'explorer l'environnement en entrant par E et cherchant la sortie S. Pensez à appeler les fonctions de services proposées dans les questions 3 et 4.

Exemple :

```
(explore etat *graph* anciensEtats capacitesPNJ)
```