Dereze Alexandra – FISA3 Informatique

Programmation fonctionnelle

Semestre 6

Professeur : Monsieur PIECHOWIAK Sylvain

I) Partie 1

a) L'énoncé

Prolog dispose de structures. Par exemple, on peut représenter une date par un triplet comprenant le jour dans le mois, le mois et l'année :

- **`date(8,4,1997).`**

A partir de cette structure on, peut facilement obtenir le jour, le mois ou l'année pour une date donnée :

- **`jour(date(X,Y,Z),X).`**
- **`mois(date(X,Y,Z),Y).`**
- **`année(date(X,Y,Z),Z).`**

On souhaite gérer les emplois du temps d'un collège à l'aide d'une base de données en PROLOG. Les séances ont toutes une durée de 2h. Les créneaux sur une journée sont 8h-10h (c1), 10h-12h (c2), 14h-16h (c3) et 16h-18h (c4). Les emplois du temps étant reproduits de semaine en semaine, chaque semaine peut être représentée par une liste de créneaux : c11, c12, c13, c14 (lundi), c21, c22, c23, c24 (mardi), c31, c22, c23, c24 (mercredi), c41, c42, c43, c44 (jeudi), c51, c52, c53, c54 (vendredi), c61, c62, c63 et c64 (samedi).

On définit les entités suivantes :

```
professeur(nom, prenom, discipline, identifiant)
eleve(nom, prenom, niveau, identifiant)
salle(nom, capacite, type , identifiant)
matiere(nom, discipline, type, niveau, identifiant)
groupe(nom, niveau, liste-eleves, identifiant)
seance(matiere, prof, groupe, salle, creneau, identifiant).
```

Les identifiants sont uniques : ce sont des entiers.

En PROLOG, il existe des prédicats qui permettent d'ajouter ou de retirer de manière dynamique des faits/prédicats dans une base : **`assert`** et **`retract`** (il en existe d'autres qui leur ressemblent et qui sont décrits dans la documentation de SWI-PROLOG).
Par exemple **`assert(salle(e215, 30, tp, 10)).`** ajoute le fait **`salle(e215, 30, tp, 10)`** dans la base. Pour le retirer, il suffit d'utiliser **`retract(salle(e215, 30, tp, 10)).`**
Tout prédicat qui donne lieu à des ajouts ou des retraits doit être déclaré comme dynamique en début de programme grâce à l'instruction :
`:- dynamic(<nom du prédicat>/<arité du prédicat>).`
Dans notre exemple, il faudra donc ajouter la ligne au début du programme
`:- dynamic(salle/4).`

1. Définir les prédicats qui permettent d'ajouter et de retirer un professeur, un élève, une salle, une matière, un groupe ou une séance. Avant d'ajouter, on s'assurera que la donnée n'est pas déjà présente dans la base et que l'identifiant n'est pas déjà utilisé.

2. Définir un prédicat qui ajoute un élève dans un groupe. On supposera qu'un élève ne peut appartenir qu'à un seul groupe.

3. Définir un prédicat qui recherche les séances qui ont un problème de capacité de salle (nombre d'élèves du groupe concerné supérieur à la capacité de la salle).

4. Définir les prédicats qui permettent d'avoir la liste des séances d'un professeur, d'un élève, d'une salle, d'un groupe.

5. Définir un prédicat qui détermine le jour associé à un créneau donné. Par exemple, ce prédicat doit renvoyer mardi pour le créneau c24.

6. Définir un prédicat qui détermine l'horaire d'un créneau donné. Par exemple, ce prédicat doit renvoyer 14h-16h pour le créneau c24

7. Définir un prédicat qui vérifie que 2 séances ne sont pas en conflit. On dira que 2 séances sont en conflit si une ressource (professeur, groupe ou salle) est présente dans les 2 séances et que les 2 séances ont lieu pendant le même créneau.

8. Définir les prédicats qui affichent l'emploi du temps d'un professeur, d'un élève, d'une salle. Pour un professeur, on affichera le nom du professeur, puis la liste des séances sous la forme (jour de la semaine, créneau horaire, matière, nom de la salle, nom du groupe). Pour un élève on affichera le nom de l'élève, puis la liste des séances sous la forme (jour de la semaine, créneau horaire, matière, nom de la salle, nom du professeur). Pour une salle on affichera le nom de la salle, puis la liste des séances sous la forme (jour de la semaine, créneau horaire, matière, nom du groupe, nom du professeur). Les prédicats **`write`** et **`writeln`** peuvent s'avérer utiles.

9. Définir les prédicats qui calculent la charge de travail d'un professeur, d'un élève, d'une salle.

On décide de représenter les classes par :

`classe(nom, niveau, groupe, liste_professeurs, liste_matieres, liste_seances, identifiant)`

Ceci permet au directeur du collège de définir les activités qu'il va devoir planifier pour l'année scolaire. Par exemple, on peut supposer qu'il y a 4 classes de 6ème, 3 classes de 5ème, 3 classes de 4ème et 2 classes de 3ème.

10. Définir les prédicats qui permettent d'ajouter et de retirer une classe. Avant d'ajouter, on s'assurera que la classe n'est pas déjà présente dans la base et que l'identifiant n'est pas déjà utilisé.

11. En supposant que chaque matière d'une classe donne lieu à 2 séances, définir un prédicat qui vérifie que toutes les séances d'une classe ont bien été planifiées.

12. Définir un prédicat qui permet au directeur du collège de s'assurer que tous ses emplois du temps sont correctement planifiés, c'est-à-dire que toutes les matières de toutes les classes sont bien planifiées et il n'y a aucun conflit. Pour cela on supposera qu'un collège est représenté par la liste de ses professeurs, de ses salles, de ses élèves, de ses groupes, de ses classes

13. Afin de conserver les données dans un fichier, prévoir les prédicats de lecture et de sauvegarde dans un fichier. Pour vous aider, une recherche sur le net peut être utile.

b) Le compte-rendu

Pour l'ensemble de ce TP, j'ai utilisé Visual Studio Code avec la librairie PySwip (Python) qui permet de coder le langage Prolog. Il aura été nécessaire d'installer SWI-Prolog et importer la librairie.

L'ensemble des captures d'écran de ce rapport seront pris d'un fichier .txt afin d'y avoir seulement le langage Prolog sans langage Python.

1) On nous demande de définir les prédicats qui permettent d'ajouter et de retirer un professeur, un élève, une salle, une matière, un groupe ou une séance. Avant d'ajouter chaque élément, il faudra s'assurer préalablement que la donnée n'est pas déjà présente dans la base et que l'identifiant n'est pas déjà utilisé.

   Ainsi pour l'ajout d'un professeur, on aura :

```
ajouter_professeur(Nom, Prenom, Discipline, Id):- not(professeur(Nom, Prenom, Discipline, _)),
                                                  not(professeur(_, _, _, Id)),
                                                  assert(professeur(Nom, Prenom, Discipline, Id))
```

FIGURE 1 : SCRIPT PERMETTANT L'AJOUT D'UN PROFESSEUR

   Avant de supprimer un élément, il faudra au préalablement vérifier que la donnée existe dans la base.

```
supprimer_professeur(Nom, Prenom, Discipline, Id):- professeur(Nom, Prenom, Discipline, Id),
                                                    retract(professeur(Nom, Prenom, Discipline, Id))
```

FIGURE 2 : SCRIPT PERMETTANT LA SUPPRESSION D'UN PROFESSEUR

   Tests :

```
ajouterProfesseur(professeur("Dupont", "Francois", "Philosophie", 1)).
ajouterProfesseur(professeur("Lefebre", "Jacques", "Maths", 1)).
ajouterProfesseur(professeur("Dupont ", "Francois", "Philosophie", 3)).
```

   Résultats :

```
True
False
False
```

2) Pour ajouter un élève à un groupe, on va définir un prédicat qui va premièrement vérifier que l'élève et le groupe se trouvent dans la base. Ensuite, on va supprimer le groupe puis le re créé en lui ajoutant l'élève dans la liste d'élève du groupe.

```
ajouter_eleve_groupe(eleve(NomEleve, PrenomEleve, NiveauEleve, IdEleve), groupe(NomGroupe, NiveauGroupe, ListeEleve, IdGroupe)):- eleve(_, _, _, IdEleve),
                                                                                                                                  groupe(_, _, _, IdGroupe),
                                                                                                                                  supprimer_groupe(NomGroupe, NiveauGroupe, ListeEleveGroupe, IdGroupe),
                                                                                                                                  ajouter_groupe(NomGroupe, NiveauGroupe, [eleve( , , , IdEleve)|ListeEleveGroupe], IdGroupe
```

FIGURE 3 : SCRIPT PERMETTANT L'AJOUT D'UN ELEVE DANS UN GROUPE

Tests :

```
ajouterEleve(eleve("Dereze", "Alex", "L3", 1)).
ajouterGroupe(groupe("109", "L1", [], 1)).
ajouter_eleve_groupe(eleve("Dereze", "Alex", "L3", 1), groupe("109", "L3", [], 1))
```

```
True
True
True
```

3) Pour rechercher les séances qui ont un problème de capacité de salle, on va avoir besoin de calculer la taille de la liste des élèves du groupe (prédicat tailleListe) et comparer cette valeur avec la capacité de la salle.

```
tailleListe([], 0 )
tailleListe([_|B], Taille) :- tailleListe(B, N) , Taille is N+1

problemeCapaciteSalle(seance(_, _, groupe(_, _, ListeEleves, _), salle(_, Capacite, _, _), _, _)) :- tailleListe(ListeEleves, TailleGroupe),
                                                                                                      TailleGroupe > Capacite
```

FIGURE 4 : SCRIPT PERMETTANT LA RECHERCHE DES SEANCES QUI ONT UN PROBLEME DE CAPACITE DE SALLE

Tests :

```
problemeCapaciteSalle(seance(_, _, groupe("carpeau", "CP", [1], 1), salle("Salle 110", 3, "TD", 1), _, _))
problemeCapaciteSalle(seance(_, _, groupe("carpeau", "CP", [1], 1), salle("Salle 110", 0, "TD", 1), _, _))
```

```
True
False
```

4) On veut pouvoir afficher la liste des séances d'un professeur, d'un élève, d'une salle, d'un groupe. Je vais prendre l'exemple de l'affichage de la liste des séances d'un élève. Pour afficher les séances d'un élève, il va falloir récupérer le groupe d'élèves participant à la séance et vérifier que l'élève recherché appartient à cette liste (prédicat appartient).

```
appartient(X, [X|_])
appartient(X,[_|L]) :- appartient(X, L)

afficher_seance_eleve(Eleve) :- seance(_, _, groupe(_, _, ListeEleves, _), _, _, Id),
                                appartient(Eleve, ListeEleves)
```

FIGURE 5 : SCRIPT PERMETTANT L'AFFICHAGE DES SEANCES D'UN ELEVE.

5) On veut définir un prédicat qui détermine le jour associé à un créneau donné. On sait que les créneaux sont écrits de la forme c<numéro du jour de la semaine><numéro pour l'heure>. Ainsi, par rapport au premier chiffre, on saura le jour de la semaine, et par rapport au deuxième chiffre, on aura les heures. Ainsi, je vais vérifier les 2 chiffres à l'aide du prédicat sub_atom qui se forme de la façon suivante :

sub_atom(chaine de caractère à analyser, Point de départ de la chaine, la longueur à considérer, La taille de la chaine après coupure, valeur à trouver).

Donc, pour l'exemple du jour mardi, on aura comme chaine à analyser : le créneau, ensuite le point de départ de la chaine sera 1 (la chaine commence à 0), puis la longueur à analyser sera aussi de 1, la taille de la chaine après coupure sera 1, et la valeur à trouver est 2.

```
jour(Creneau) :- sub_atom(Creneau, 1, 1, 1, 2), write("Mardi")
```

FIGURE 6 : SCRIPT PERMETTANT DE DETERMINER LE JOUR ASSOCIE A UN CRENEAU

6) On va utiliser le même prédicat pour déterminer l'horaire d'un créneau donné.

Pour le créneau 16h-18h, il faudra vérifier que le deuxième chiffre soit 4, on aura comme chaine à analyser : le créneau, ensuite le point de départ de la chaine sera 2 (la chaine commence à 0), puis la longueur à analyser sera aussi de 1, la taille de la chaine après coupure est 0, et la valeur à trouver est 4.

```
heure(Creneau) :- sub_atom(Creneau, 2, 1, 0, 4), writeln("16h-18h")
```

FIGURE 7 : SCRIPT PERMETTANT DE DETERMINER L'HEURE ASSOCIEE A UN CRENEAU

7) On veut définir un prédicat qui vérifie que 2 séances ne sont pas en conflit. Pour vérifier que 2 séances ne sont pas en conflits, il va falloir vérifier que le professeur, le groupe ou la salle sont différents et que les créneaux des séances sont aussi différents.

```
conflit(seance(_, Professeur, _, _, Creneau, _), seance(_, Professeur, _, _, Creneau, _)) :- write("Séance en conflit : Les professeurs sont identiques")
conflit(seance(_, _, Groupe, _, Creneau, _), seance(_, _, Groupe, _, Creneau, _)) :- write("Séance en conflit : Les groupes sont identiques")
conflit(seance(_, _, _, Salle, Creneau, _), seance(_, _, _, Salle, Creneau, _)) :- write("Séance en conflit : Les salles sont identiques")
```

FIGURE 8 : SCRIPT PERMETTANT DE DETERMINER LE JOUR ASSOCIE A UN CRENEAU

Tests :

conflit(seance(matiere("Francais", "Francais", "Litteraire", "CE1", 1), professeur("Dupont", "Francois", "Francais", 1), groupe("carpeau", "CP", [], 1), salle("110", 10, "cours", 1), "c21", 1), seance(matiere("Maths", "Maths", "Maths", "CE2", 1), professeur("Lefebre", "Jacques", "Maths", 2), groupe("franki", "CM2", [], 2), salle("109", 40, "cours", 1), "c23", 2))

conflit(seance(matiere("Francais", "Francais", "Litteraire", "CE1", 1), professeur("Dupont", "Francois", "Francais", 1), groupe("carpeau", "CP", [], 1), salle("110", 10, "cours", 1), "c21", 1), seance(matiere("Maths", "Maths", "Maths", "CE2", 1), professeur("Lefebre", "Jacques", "Maths", 2), groupe("franki", "CM2", [], 2), salle("110", 40, "cours", 1), "c23", 2))

```
True
False
```

8) On veut définir les prédicats affichent l'emploi du temps d'un professeur, d'un élève, d'une salle. Ainsi, je vais prendre l'exemple de l'affichage de l'emploi du temps d'un professeur. On va d'abord afficher son nom puis la liste des séances auquel il appartient. On affichera le jour de la semaine suivi du créneau horaire, la matière et nom de la salle puis le nom du groupe. On va d'abord vérifier les séances auquel le groupe appartient, puis à partir de là, on va pouvoir récupérer les informations des séances nécessaires (créneau, matière, salle et groupe).

```
afficherEmploiDuTemps(professeur(NomProfesseur, Prenom, Discipline, Id)) :- seance(Matiere, professeur(NomProfesseur, Prenom, Discipline, Id), groupe(NomGroupe, _, _,
                                                                            salle(NomSalle, _, _, _), Creneau, Id),
                                                                            writeln(NomProfesseur),
                                                                            writeln(jour(Creneau)),
                                                                            writeln(heure(Creneau)),
                                                                            writeln(Matiere),
                                                                            writeln(NomSalle),
                                                                            writeln(NomGroupe)')
```

FIGURE 9 : SCRIPT PERMETTANT DE DONNER L'EMPLOI DU TEMPS D'UN PROFESSEUR

9) On veut définir les prédicats qui calculent la charge de travail d'un professeur, d'un élève, d'une salle. Ainsi, je vais prendre l'exemple pour un professeur. On va parcourir l'ensemble des séances contenant ce professeur et on va les compter avec le prédicat count fourni par Prolog. Puis sachant que chaque séance dure 2heure, on va multiplier par 2 le résultat obtenu et on obtient la charge de travail d'un professeur.

```
chargeDeTravail(professeur(Nom, Prenom, Discipline, Identifiant)) :- aggregate_all(count, seance(_, professeur(Nom, Prenom, Discipline, Identifiant), _, _, _, _), TotalSeance),
Total is 2*TotalSeance,
write("Charge de travail du professeur : "),
write(Total)
```

FIGURE 10 : SCRIPT PERMETTANT DE CALCULER LA CHARGE DE TRAVAIL D'UN PROFESSEUR

10) On nous demande de définir les prédicats qui permettent d'ajouter et de retirer une classe. Avant d'ajouter chaque élément, il faudra s'assurer préalablement que la donnée n'est pas déjà présente dans la base et que l'identifiant n'est pas déjà utilisé.

Ainsi pour l'ajout, on aura :

```
ajouter_classe(Nom, Niveau, Groupe, Liste_professeurs, Liste_matieres,Liste_seances, Id) :- not(classe(Nom, Niveau, Groupe, Liste_professeurs, Liste_matieres, Liste_seances, _)
                                                                                            not(classe(_, _, _,_,_,_, Id)), assert(classe(Nom, Niveau, Groupe, Liste_professeurs
                                                                                            Liste_matieres,Liste_seances, Id))
```

FIGURE 11 : SCRIPT PERMETTANT L'AJOUT D'UN PROFESSEUR

Avant de supprimer un élément, il faudra au préalablement vérifier que la donnée existe dans la base.

```
supprimer_classe(Nom, Niveau, Groupe, Liste_professeurs, Liste_matieres,Liste_seances, Id):- classe(Nom, Niveau, Groupe, Liste_professeurs, Liste_matieres,Liste_seances, Id),
                                                                                              retract(classe(Nom, Niveau, Groupe, Liste_professeurs, Liste_matieres,Liste_seances, Id))
```

FIGURE 12 : SCRIPT PERMETTANT LA SUPPRESSION D'UN PROFESSEUR

11) On doit définir un prédicat qui vérifie que toutes les séances d'une classe ont bien été planifiées. Pour vérifier que chaque séance a bien été planifiée, on va dans un premier temps récupérer la taille des listes de matières puis de séances et on va les comparer. Sachant qu'une matière équivaut à 2 séances, il doit y avoir deux fois plus de séance que de matière. Sinon, on peut en déduire que les séances n'ont pas été planifiées.

Dans un second temps, on va vérifier que les matières des séances sont bien présentes que 2 fois par classe. Ainsi, il va falloir parcourir la liste des séances et la liste des matières et on va compter le nombre de fois où apparait chaque matière. Si une matière ne possède pas 2 séances, il y a un problème dans la planification.

```
verificationSeances(classe(_, _, _, _, ListeMatieres, ListeSeances, _)) :- taille(ListeMatieres, TailleListeMatieres),
                                                                           taille(ListeSeances, TailleListeSeances),
                                                                           TailleListeSeances = 2*TailleListeMatieres, verificationSeancesCorrespondantes(ListeSeances, ListeMatieres).

verificationSeancesCorrespondantes(_, []).

verificationSeancesCorrespondantes(ListeSeances, [Matiere|ListeMatieres]) :-  aggregate_all(count, matiereAppartient(Matiere, ListeSeances), Total
                                                                               Total = 2,
                                                                               verificationSeancesCorrespondantes(ListeSeances, ListeMatieres).

matiereAppartient(matiere, [seance(matiere, _, _, _, _, _)|_]).
matiereAppartient(matiere, [_|L]) :- matiereAppartient(matiere, L).
```

FIGURE 13 : SCRIPT PERMETTANT DE DETERMINER SI TOUTES LES SEANCES ONT BIEN ETE PLANIFIEES

12) On doit définir un prédicat qui vérifie que les emplois du temps sont bien planifiés. Pour vérifier que les emplois du temps sont bien planifiés, il va falloir parcourir l'ensemble des classes. On va d'abord vérifier que la classe existe bien puis on va appeler le prédicat précédent afin de vérifier que toutes les séances ont bien été planifiée.

```
emploiDuTempsPlanifie(_, []).
emploiDuTempsPlanifie([classe(Nom, Niveau, Groupe, ListeProfesseur, ListeMatieres, ListeSeances, Identifiant)|ListeClasses]) :- classe(Nom, Niveau, Groupe, ListeProfesseur, ListeMatieres, ListeSeances, Identifiant),
                                                                                                                                verificationSeances(classe(Nom, Niveau, Groupe, ListeProfesseur, ListeMatieres, ListeSeances, Identifiant)),
                                                                                                                                write("Classe : "),
                                                                                                                                write(Nom),
                                                                                                                                writeln(" planification ok"),
                                                                                                                                emploiDuTempsPlanifie(ListeClasses).
```

FIGURE 14 : SCRIPT PERMETTANT DE DETERMINER SI LES EMPLOIS DU TEMPS SONT BIEN PLANIFIES

13) On veut définir le prédicat permettant de sauvegarder les données. On va utiliser les prédicats open, with_output_to, close fourni par PROLOG.

Open s'utilise de la façon suivante : *open(Fichier, Statut de l'ouverture de fichier, Flux)*

With_output_to s'utilise : *with_output_to(Flux, les termes à écrire)*

Close s'utilise : *close(Flux)*

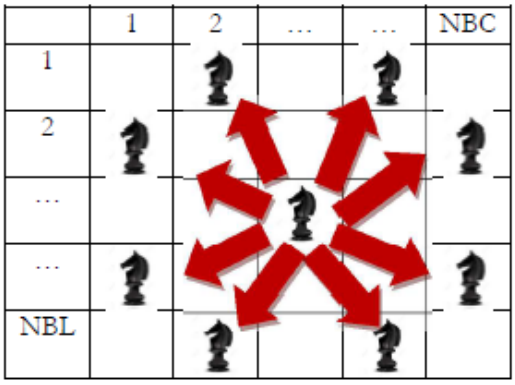
```
sauvegarder(Fichier) :- open(Fichier, write, Flux), with_output_to(Flux, listing), close(Flux).
```

FIGURE 15 : SCRIPT PERMETTANT DE SAUVEGARDER LES DONNEES

II) Partie 2

a) L'énoncé

On souhaite résoudre le problème du cavalier. Sur un échiquier on place un cavalier et on cherche à savoir s'il peut se déplacer sur toutes les cases sans jamais repasser plusieurs fois par la même. Les mouvements possibles d'un cavalier sont décrits dans le schéma suivant.

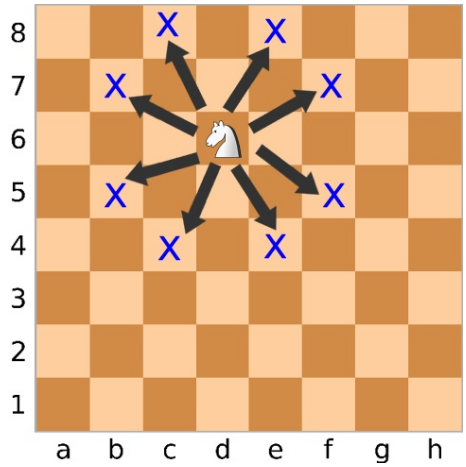


On suppose que la taille de l'échiquier est NBL × NBC.

1. Pouvez-vous estimer le nombre de déplacements d'un cavalier en fonction de la taille de l'échiquier ?
2. Proposez une solution en PROLOG pour ce problème.
3. On modifie le problème et cette fois on place 2 cavaliers sur l'échiquier. Chaque cavalier se déplace à tour de rôle et ne peut se déplacer que sur une case qui n'a pas été occupée.
   Proposez une solution en PROLOG pour ce nouveau problème.

b) Le compte-rendu

1) Un cavalier pour rappel ne peut se déplacer que en L.



Ainsi, tant que la taille du plateau est inférieure ou égale à 2, il ne peut pas se déplacer.

Sur un plateau 3x3, après plusieurs essais, un cavalier peut se déplacer 8 fois.

Sur un plateau 4x4, après plusieurs essais, un cavalier peut se déplacer 15 fois.

Sur un plateau 4x3, après plusieurs essais, un cavalier peut se déplacer 11 fois.

J'en déduis qu'un cavalier sur un plateau NBCxNBL peut se déplacer de NBCxNBL-1.