École Centrale Lyon

Algorithme et raisonnement
PROLOG
Rapport

BE2 - Recettes

Élèves :
Hugo Puybareau

Enseignant :
Emmanuel Dellandrea

20 mars 2024

Table des matières

1 Introduction

Dans ce travail à rendre, il nous est donné une base de données *'bd_recettes1.pl'*. L'objectif du travail est de créer des prédicats et des relations ainsi que de tester ces derniers pour pouvoir manipuler tout un ensemble de recettes.

La base de données, dans son état initial, est la suivante :

```
%predicat disponible(ingrédient)
disponible(eau).
disponible(sucre).
disponible(sel).
disponible(vinaigre).
disponible(huile).
disponible(beurre).
disponible(sachet_de_the).
disponible(cafe).
disponible(chocolat).
disponible(lait).
disponible(farine).
disponible(oeufs).
disponible(pate_a_crepes).
disponible(pates_nouilles).
disponible(laitue).
disponible(pommes_dt).
disponible(oignon).
disponible(tomates).
disponible(creme).
disponible(fromage).    % toutes s
disponible(lardons).    % pour tes

%prédicat recette(mets, liste des ingrédients)
recette(expresso, [cafe, eau, sucre ]).
recette(gateau, [lait, farine, oeufs, sucre, beurre ]).
recette(the, [sachet_de_the, eau, sucre ]).
recette(crepes, [pate_a_crepes, sucre, beurre ]).
recette(salade, [laitue, vinaigre, huile,  oignon]).
recette(puree, [lait, pommes_dt, beurre,  sel ]).
recette(frites, [pommes_dt, huile,  sel]).
recette(pates_a_la_carbonara, [pates_nouilles,  oeufs,  lardons, sel, oignon, creme]).
recette(carbona, [pates_nouilles,  oeufs,  lardons,  sel, oignon ]).
recette(pitza, [farine,  oeufs,  lardons,  sel, oignon, tomates]).
recette(tartiflette, [pommes_dt,  lardons,  sel, fromage]).
```

FIGURE 1 – Base de données fournie

2 Question 1

Dans la question 1, il nous est demandé de créer la relation *'peut_preparer(R)'*.

Je code de la manière suivante en m'inspirant du travail qui est fait dans le TD sur les têtes/restes de liste :

```
% Vérifie si on peut préparer un mets R en vérifiant si tous les ingrédients nécessaires sont disponibles.
peut_preparer(R) :-
    recette(R, Ingredients),
    tous_disponibles(Ingredients).

% Vérifie si tous les ingrédients d'une liste sont disponibles
tous_disponibles([]). % Condition de base : si la liste est vide, c'est vrai.
tous_disponibles([Tete|Reste]) :- % Décompose la liste en tête et reste
    disponible(Tete), % Vérifie si la tête est disponible
    tous_disponibles(Reste). % Récursion sur le reste
```

FIGURE 2 – Code pour la question 1

Pour les tests, j'enlève la disponibilité de l'eau :

```
?- disponible(eau).
false.

?- peut_preparer(expresso).
false.

?- peut_preparer(gateau).
true.
```

FIGURE 3 – Test pour la question 1

3 Question 2

Dans la question 2, il nous est demandé de créer la relation *'a_ besoin_ de(R, I)'*.
Pour ce faire, on code :

```
%a_besoin_de

membre(Element, [Element|_]).
membre(Element, [_|Reste]) :-
    membre(Element, Reste).

a_besoin_de(Recette, Ingredient) :-
    recette(Recette, Liste),
    membre(Ingredient, Liste).
```

FIGURE 4 – Code pour la question 2

On teste avec les tests suivants :

```
?- a_besoin_de(gateau, lait).
true .

?- a_besoin_de(gateau, sel).
false.
```

FIGURE 5 – Test pour la question 2

4 Introduction des quantités

On doit recoder les prédicats des deux premières questions en prennant en compte les quantités :

Pour le prédicat *'peut_ preparer(R)'* :

```
% Vérifie si on peut préparer un mets R en vérifiant si tous les ingrédients nécessaires sont disponib
les.
peut_preparer(R) :-
    recette(R, Ingredients),
    tous_disponibles(Ingredients).

% Vérifie si tous les ingrédients d'une liste sont disponibles en quantité suffisante
tous_disponibles([]). % Condition de base : si la liste est vide, c'est vrai.
tous_disponibles([(Ingredient, QuantiteNecessaire)|Reste]) :-
    disponible(Ingredient, QuantiteDisponible),
    QuantiteDisponible >= QuantiteNecessaire,
    tous_disponibles(Reste).
```

FIGURE 6 – Code de *'peut_preparer(R)'* avec les quantités

```
?- disponible(eau, A).
A = 50.

?- peut_preparer(expresso).
true.

% /Users/hugopuybareau/Documents/Centrale/INF/PROLOG/DM2_PUYBAREAU/bd_recettes2
compiled 0.00 sec, 0 clauses
?- disponible(eau, A).
A = 1.

?- peut_preparer(expresso).
false.
```

FIGURE 7 – Test de *'peut_preparer(R)'* avec les quantités

On teste en modifiant la quantité sur l'eau :
Pour le test avec 'peut_preparer(X)', j'obtiens :

```
?- peut_preparer(X).
X = expresso ;
X = gateau ;
X = the ;
X = crepes ;
X = salade ;
X = puree ;
X = frites ;
X = pates_a_la_carbonara ;
X = carbona ;
X = pitza ;
X = tartiflette.
```

FIGURE 8 – Toutes les recettes qu'il est possible de préparer avec les quantités fournies dans le code

5 Gestion des quantités

Pour la méthode 1, comme expliqué dans l'énoncé, on utilise une liste qui contient tous les ingrédients propres à une recette ainsi que leur quantités.

Je n'ai ensuite pas réussi à réaliser la méthode comme décrite dans l'énoncé. Pour modifier la base de données, j'ai trouvé sur internet qu'il était possible de définir le prédicat *'disponibe(R, Q)'* comme *'dynamic'* en ajoutant la ligne : *:- dynamic disponible/2* . Cette déclaration informe Prolog que le prédicat disponible/2 peut être modifié dynamiquement au cours de l'exécution du programme, permettant ainsi l'utilisation de *assert/1* et *retract/1*.

Les méthodes *'assert'* et *'retract'* permettent d'ajouter ou de retirer un fait ou une règle d'une base de données. Leur utilisation est assez simple à comprendre en regardant mon code :

```
% Vérifie si on peut préparer un mets R en vérifiant si tous les ingrédients nécessaires sont disponibles.
peut_preparer(R, Restants) :-
    recette(R, Ingredients),
    tous_disponibles(Ingredients, Restants).

% Vérifie si tous les ingrédients d'une liste sont disponibles en quantité suffisante
tous_disponibles([], []).
tous_disponibles([(Ingredient, QuantiteNecessaire) | Reste], [(Ingredient, NouvelleQuantite) | Restants]) :-
    disponible(Ingredient, QuantiteDisponible),
    QuantiteDisponible >= QuantiteNecessaire,
    NouvelleQuantite is QuantiteDisponible - QuantiteNecessaire,
    tous_disponibles(Reste, Restants).

% Met à jour la base de données avec les nouvelles quantités
mettre_a_jour_quantites([]).
mettre_a_jour_quantites([(Ingredient, NouvelleQuantite) | Reste]) :-
    retract(disponible(Ingredient, _)),
    assert(disponible(Ingredient, NouvelleQuantite)),
    mettre_a_jour_quantites(Reste).
```

FIGURE 9 – Code pour la gestion des quantités

Attention cela ne veut pas dire que la base de données dans mon script et modifiée mais uniquement celle que j'exécute dans Prolog.

Pour le test, je fais :

```
?- peut_preparer(expresso, Restants), mettre_a_jour_quantites(Restants).
Restants = [(cafe, 7), (eau, 40), (sucre, 99)].

?- disponible(cafe,R1), disponible(eau, R2), disponible(sucre, R3).
R1 = 7,
R2 = 40,
R3 = 99.

?- peut_preparer(expresso, Restants), mettre_a_jour_quantites(Restants).
Restants = [(cafe, 4), (eau, 30), (sucre, 98)].

?- disponible(cafe,R1), disponible(eau, R2), disponible(sucre, R3).
R1 = 4,
R2 = 30,
R3 = 98.
```

FIGURE 10 – Test pour la gestion des quantités

6 Bonus : Livres

6.1 Même Nom/Prenom

```
?- livres(auteur(P1,N1), _), livres(auteur(P2,N2), _), P1=P2, N1\=N2.
P1 = P2, P2 = 'Jacques',
N1 = 'Lacan',
N2 = 'Derrida' ;
P1 = P2, P2 = 'Michel',
N1 = 'Foucault',
N2 = 'Serres' ;
P1 = P2, P2 = 'Jacques',
N1 = 'Derrida',
N2 = 'Lacan' ;
P1 = P2, P2 = 'Michel',
N1 = 'Serres',
N2 = 'Foucault' ;
P1 = P2, P2 = 'Simone',
N1 = 'Weil',
N2 = 'De Beauvoir' ;
P1 = P2, P2 = 'Simone',
N1 = 'De Beauvoir',
N2 = 'Weil' ;
false.

?- livres(auteur(P1,N1), _), livres(auteur(P2,N2), _), N1=N2, P1\=P2.
false.
```

FIGURE 11 – Test pour les noms et prenoms

6.2 Livres d'un auteur dont le prix est inférieur/supérieur/égal à un prix donné

Pour cette partie j'ai rédigé le code suivant :

```
% Trouver et afficher les livres d'un auteur dont le prix est inférieur à un certain prix
livre_prix_min(Prenom, Nom, Prix) :-
    livres(auteur(Prenom, Nom), Livres),
    filtrer_livres_prix_inf(Livres, Prix, Resultats),
    afficher_livres(Resultats).

% Filtrer les livres dont le prix est inférieur à Prix
filtrer_livres_prix_inf([], _, []).
filtrer_livres_prix_inf([(Titre, P)|Reste], Prix, [Titre|Filtres]) :-
    P < Prix,
    filtrer_livres_prix_inf(Reste, Prix, Filtres).
filtrer_livres_prix_inf([(_, P)|Reste], Prix, Filtres) :-
    P >= Prix,
    filtrer_livres_prix_inf(Reste, Prix, Filtres).

% Afficher les titres des livres filtrés
afficher_livres([]).
afficher_livres([Titre|Reste]) :-
    write(Titre), nl,
    afficher_livres(Reste).
```

FIGURE 12 – Code pour la partie la question sur les prix minimums

Je fais une fonction principale qui appelle deux autres fonctions. Pour trouver les livres je construis une liste *'Filtres'* par appel récursif. Puis, pour afficher les titres contenus dans la liste j'utilise *'write'* et *'nl'* que j'ai trouvé sur internet.

Pour le test, j'ai utilisé Simone Weil :

```
?- livre_prix_min('Simone', 'Weil', 27).
Attente de Dieu
Cahiers
```

FIGURE 13 – Test sur Simone Weil

Pour le prix supérieur et égal, il faut juste changer les conditions dans la fonction *'filtrer_livres_prix_inf'*.