FEUP-L.EIC024-2024/2025-1S

Início	quinta-feira, 26 de dezembro de 2024 às 12:05
Estado	Prova submetida
Data de submissão:	quinta-feira, 26 de dezembro de 2024 às 14:05
Tempo gasto	2 horas

Informação

Read these instructions carefully before you start the test.

- This is an open-book test; you can use the materials available on the local computer and Moodle's course page, but no printed materials are allowed.
- You can use the resources on the computer, such as SICStus Prolog (and its manual).
- **The presence of electronic and/or communication devices is strictly forbidden.**
- The maximum duration of the test is stated below.
- The score of each question is stated in the test, totaling 20 points.
- A wrong answer in a multiple-choice question with four options implies a penalty equal to 25% of the question's value.
- A wrong answer in a True/False question implies a penalty equal to 50% of the question's value.
- Fraud attempts will be punished by the annulment of the test for all intervenients.

- **Use the predicate names and argument order exactly as specified in the test statement.**
- **Do not copy any code provided in the questions into the answer text box.**
- **You can use predicates requested in previous questions even if you haven't implemented them.**
- **However, do not include answers to previous questions in the current one.**
- **If you implement auxiliary predicates, you must include them in the answers to all questions where they are used.**
- **Pay attention to syntactic errors (such as forgetting the '.' after each rule/fact).**
- **Document your code and use intuitive variable names to clarify code interpretation.**

Be mindful of the time available.

Good work!

Informação

Consider the following knowledge base about the *Pasta & Flour Lounge* (PFL) restaurant.

Each *dish/3* fact contains a dish's name, the price for which it is sold in the restaurant, and a list with the quantity of each ingredient needed to produce it (each ingredient appears only once on the list). For instance, to produce a pizza, which is sold for 2200 cents, 300g of cheese and 350g of tomato are needed.

Each *ingredient/2* fact contains an ingredient's name and unit cost (i.e., how many cents are required to buy 1 gram).

```
% dish(Name, Price, IngredientGrams).
dish(pizza,         2200, [cheese-300, tomato-350]).
dish(ratatouille,   2200, [tomato-70, eggplant-150, garlic-50]).
dish(garlic_bread,  1600, [cheese-50, garlic-200]).

:- dynamic ingredient/2.

% ingredient(Name, CostPerGram).
ingredient(cheese,   4).
ingredient(tomato,   2).
ingredient(eggplant, 7).
ingredient(garlic,   6).
```

Answer questions 1 to 7 **WITHOUT** using multiple solution predicates (findall, setof, and bagof), and **WITHOUT** using any SICStus library.

Pergunta 1

Pontuação 1,000

Implement *count_ingredients(?Dish, ?NumIngredients)*, which determines how many different ingredients are needed to produce a dish.

```
count_ingredients(Dish, N):-
        dish(Dish, _Cost, L),
        length(L, N).
```

Pergunta 2

Pontuação 1,000

Implement *ingredient_amount_cost(?Ingredient, +Grams, ?TotalCost)*, which determines the total cost (in cents) of buying a certain amount (in grams) of an ingredient.

```
ingredient_amount_cost(Ingredient, Grams, Cost):-
        ingredient(Ingredient, UnitCost),
        Cost is Grams * UnitCost.
```

Pergunta 3

Pontuação 1,250

Implement *dish_profit(?Dish, ?Profit)*, which determines the profit of selling a dish in the restaurant. A dish's profit is the difference between its price and the combined cost of its ingredients.

```
dish_profit(Dish, Profit):-
        dish(Dish, Price, L),
        dish_cost(L, TotalCost),
        Profit is Price - TotalCost.

dish_cost([], 0).
dish_cost([Ingredient-Grams|Is], TotalCost):-
        ingredient_amount_cost(Ingredient, Grams, Cost),
        dish_cost(Is, TotalCost1),
        TotalCost is TotalCost1 + Cost.
```

Pergunta 4

Pontuação 1,000

Implement *update_unit_cost(+Ingredient, +NewUnitCost)*, which modifies the knowledge base by updating the unit cost of an ingredient. If the ingredient does not exist, it should be added to the knowledge base. The predicate must always succeed.

```
update_unit_cost(Ingredient, NewUnitCost):-
        retract( ingredient(Ingredient, _OldCost) ), !,
        assert( ingredient(Ingredient, NewUnitCost) ).
update_unit_cost(Ingredient, NewUnitCost):-
        assert( ingredient(Ingredient, NewUnitCost) ).
```

Pergunta 5

Pontuação 1,250

Implement *most_expensive_dish(?Dish, ?Price)*, which determines the most expensive dish one can eat at the restaurant and its price. In case of a tie, the predicate must return, via backtracking, each of the most expensive dishes.

```
most_expensive_dish(Dish, Price):-
        dish(Dish, Price, _),
        \+((
            dish(_, Price1, _),
            Price1 > Price
        )).
```

Pergunta 6

Pontuação 1,500

Implement *consume_ingredient(+IngredientStocks, +Ingredient, +Grams, ?NewIngredientStocks)*, which receives a list of ingredient stocks (as pairs of Ingredient-Amount), an ingredient, and an amount (in grams) and computes a new list obtained from removing the given amount of ingredient from the original stock. The predicate must only succeed if there is enough ingredient in stock.

Constraint: In this question, and in this question only, you are **not** allowed to use recursion. Solutions using recursion will only receive up to 25% of the maximum score.

Examples:

```
| ?- consume_ingredient( [garlic-600, tomato-800, cheese-750], tomato, 150, L).
L = [garlic-600, tomato-650, cheese-750] ? ;
no
| ?- consume_ingredient( [garlic-600, tomato-800, cheese-750], tomato, 1000, L).
no
```

```
consume_ingredient(IngredientStocks, Ingredient, Grams, NewIngredientStocks):-
        append(Prefix, [Ingredient-Quant | Suffix], IngredientStocks),
        NewQuant is Quant - Grams,
        NewQuant >= 0,
        append(Prefix, [Ingredient-NewQuant | Suffix], NewIngredientStocks).
```

Pergunta 7

Pontuação 1,500

Implement *count_dishes_with_ingredient(+Ingredient, ?N)*, which determines how many dishes use the given ingredient.

```
count_dishes_with_ingredient(Ingredient, N):-
        gather_dishes_with_ingredient(Ingredient, [], Dishes),
        length(Dishes, N).

gather_dishes_with_ingredient(Ingredient, Acc, L):-
        dish(Dish, _, Ings),
        \+member(Dish, Acc),
        member(Ingredient-_Amnt, Ings),
        !,
        gather_dishes_with_ingredient(Ingredient, [Dish|Acc], L).
gather_dishes_with_ingredient(_, L, L).
```

Informação

In the following questions, you can use multiple solution predicates (findall, setof, and bagof) as well as any SICStus library.

Pergunta 8

Pontuação 1,250

Implement *list_dishes(?DishIngredients)*, which returns a list of pairs Dish-ListOfIngredients.

Example:

```
| ?- list_dishes(L).
L = [pizza-[cheese, tomato], ratatouille-[tomato, eggplant, garlic], garlic_bread-[cheese, garlic]] ? ;
no
```

```
list_dishes(DishIngredients):-
        findall(D-L, ( dish(D, _, _), list_ingredients(D, L) ), DishIngredients).

list_ingredients(Dish, Ingredients):-
        dish(Dish, _, L),
        findall(Ingredient, member(Ingredient-_Amnt, L), Ingredients).
```

Pergunta 9

Pontuação 1,250

Implement *most_lucrative_dishes(?Dishes)*, which returns the restaurant's dishes, sorted by decreasing amount of profit. In case of a tie, any order of the tied dishes will be accepted.

Example:

```
| ?- most_lucrative_dishes(L).
L = [ratatouille,pizza,garlic_bread] ? ;
no
```

```
most_lucrative_dishes(Dishes):-
        setof(Profit-Dish, dish_profit(Dish, Profit), Costs),
        reverse(Costs, CostsInv),
        findall(Dish, member(_-Dish, CostsInv), Dishes).
```

Informação

Consider the following predicates.

```
predX(S, D, NewS):-
        dish(D, _, L),
        predY(L, S, NewS).

predY([], L, L).
predY([I-Gr|Is], S, NewS):-
        !,
        consume_ingredient(S, I, Gr, S1),
        predY(Is, S1, NewS).
```

Pergunta 10

Pontuação 1,250

Complete the text below (in each hole to fill, only one option is correct):

predX/3 receives as argument (in order) a [ ] and a [ ]

to return a new list of [ ] after [ ] .

predX/3 [ ] .

Resposta correta:
Complete the text below (in each hole to fill, only one option is correct):

predX/3 receives as argument (in order) a [list of ingredient-quantity pairs] and a [dish name] to return a new list of [ingredient-quantity pairs] after [using the ingredients in stock to produce the dish].

predX/3 [fails if there aren't enough ingredients in stock].

Pergunta 11

Pontuação 0,250

The cut present in the recursive clause of *predY/3* is green. True or false?

Selecione uma opção:

- Verdadeiro
- Falso

A resposta correta é "Verdadeiro"

Informação

Consider the following predicate.

```
predZ:-
        read(X),
        X =.. [_|B],
        length(B, N),
        write(N), nl.
```

Pergunta 12

Pontuação 1,000

Explain concisely (in one sentence) what *predZ/0* does.

```
predZ/0 prompts the user the write a term and prints its arity in the console.
```

Pergunta 13

Pontuação 0,500

Which of the following statements about tail recursion is correct?

Tail recursion is a special type of recursion where the recursive call is the last thing that happens in the function.

a. By using an extra argument, one can rewrite certain recursive predicates so that they are tail-recursive.

b. Tail recursion occurs not only in rules but also in facts.

c. *predY/3* uses tail recursion due to the usage of a cut in the recursive clause.

d. *predZ/0* uses tail recursion.

e. All other statements are incorrect.

Resposta correta: By using an extra argument, one can rewrite certain recursive predicates so that they are tail-recursive.

Pergunta 14

Pontuação 0,500

Consider the following statements about difference lists.

A - The [a,b|T]\T difference list is equivalent to [a,b].
B - Difference lists provide O(1) access to an uninstantiated prefix of a list.
C - Using difference lists, we can compute the length of a list's prefix in constant time (O(1)).

Which statements are correct?

a. A, B, and C

b. Only A

c. Only C

d. A and C

e. B and C

Resposta correta: Only A

Informação

Using operators, the goal is to write facts in the following format:

```
garlic_bread requires cheese and garlic.
garlic_bread requires cheese and some garlic.
ratatouille requires tomato and eggplant and garlic.
```

Note: "some" can be used once before each ingredient.

Consider "*requires*" to be defined as follows:

```
:- op(590, xfx, requires).
```

Pergunta 15

Pontuação 0,250

Which is the most correct way of defining "*some*" in terms of syntax and semantics?

a. `:- op(570, fy, some).`

b. `:- op(600, fy, some).`

c. `:- op(570, fx, some).`

d. `:- op(600, xf, some).`

e. `:- op(570, xf, some).`

Resposta correta:

```
:- op(570, fx, some).
```

Pergunta 16

Pontuação 0,250

Which is the most correct way of defining "*and*" in terms of syntax and semantics?

a. `:- op(580, yxfx, and).`

b. `:- op(580, xfy, and).`

c. `:- op(580, yf, and).`

d. `:- op(580, xfx, and).`

e. `:- op(580, xf, and).`

Resposta correta:

```
:- op(580, xfy, and).
```

Informação

Consider the following query:

```
| ?- member(X, [a,b,c,d,e]), !, member(Y, [1,2,3,4]).
```

Pergunta 17

Pontuação 0,250

The cut present in the query is green. True or false?

Selecione uma opção:

Verdadeiro

Falso

A resposta correta é "Falso"

Pergunta 18

Pontuação 0,250

How many children does the root of the search tree of the query have?

a. 3
b. 0
c. 1
d. 2
e. 5

Resposta correta: 1

Pergunta 19

Pontuação 0,250

Backtracking: The Prolog engine backtracks to the second member predicate and unifies Y with 2, then 3, and finally 4.

How many unifications are performed, in total, when executing the query and asking for all the solutions via backtracking?

a. 20
b. 5
c. 4
d. 9
e. 1

Resposta correta: 5

Pergunta 20

Pontuação 0,250

Moving the cut to the beginning of the query decreases the number of unifications performed when executing the query. True or false?

Selecione uma opção:

Verdadeiro
Falso

A resposta correta é "Falso"

Informação

When several drugs/medicaments have the effect of healing a patient with some disease, a smart step is to determine a molecular substructure common to them all. This substructure might be a starting point for understanding the adequate treatment for that disease.

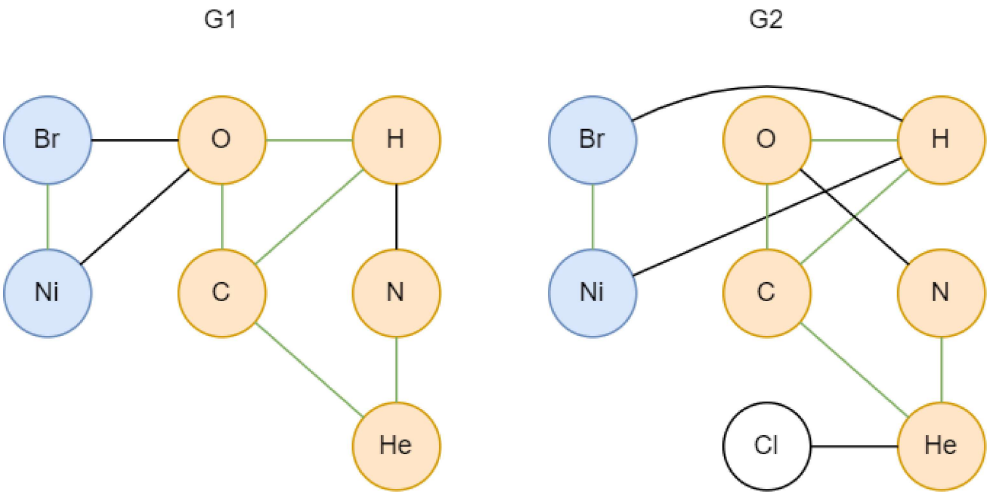
Consider that molecules are represented as undirected graphs, where atoms are the graph's vertices, and bonds are the graph's edges. One relevant procedure in this type of analysis would be determining the largest subgraph common to a given pair of molecules. Note: In each graph, the vertex names (i.e. atom names) are unique.

Consider the following examples of graphs:

```
%G1
edge(g1, br, o).
edge(g1, br, ni).
edge(g1, o, ni).
edge(g1, o, c).
edge(g1, o, h).
edge(g1, h, c).
edge(g1, h, n).
edge(g1, n, he).
edge(g1, c, he).

% G2
edge(g2, br, h).
edge(g2, br, ni).
edge(g2, h, ni).
edge(g2, h, o).
edge(g2, h, c).
edge(g2, o, c).
edge(g2, o, n).
edge(g2, n, he).
edge(g2, c, he).
edge(g2, cl, he).
```

The figure below depicts both graphs. The edges in green are the common edges of both graphs. The colored vertices denote the two common subgraphs.



Pergunta 21

Pontuação 2,000

Implement *common_edges(+G1, +G2, ?L)*, which, given the identifiers of two graphs (G1 and G2), computes their list of common edges.

Example:

```
| ?- common_edges(g1,g2,L).
L = [br-ni,o-c,o-h,h-c,n-he,c-he] ? ;
no
```

```
con(G, X-Y):-
        edge(G, X, Y).
con(G, X-Y):-
        edge(G, Y, X).

common_edges(G1, G2, Edges) :-
    findall(V1-V2, ( edge(G1, V1, V2), con(G2, V1-V2) ), Edges).
```

Pergunta 22

Pontuação 2,000

Implement *common_subgraphs(+G1, +G2, ?Subgraphs)*, which determines the list of vertices of each common subgraph of both input graphs. Any order of the subgraphs and of the vertices will be accepted.

Example:

```
| ?- common_subgraphs(g1,g2,L).
L = [[br,ni],[c,h,he,n,o]] ? ;
no
```

```
common_subgraphs(G1, G2, Subgraphs):-
        common_edges(G1, G2, Edges),
        common_subgraphs_aux(Edges, Subgraphs).

common_subgraphs_aux([], []).
common_subgraphs_aux([V1-V2|Es], [SGNoDups|SGs]):-
        next_subgraph([V1,V2], Es, NewEs, SG),
        sort(SG, SGNoDups), % remove duplicate nodes
        common_subgraphs_aux(NewEs, SGs).

adjacent(V, V-_).
adjacent(V, _-V).

next_subgraph(Vs, Es, Es, Vs):-
        \+((
            member(V, Vs),
            select(E, Es, _),
            adjacent(V, E)
        )), !.
next_subgraph(Vs, Es, NewEs, SG):-
        member(V, Vs),
        select(V1-V2, Es, Es1),
        adjacent(V, V1-V2),
        !,
        next_subgraph([V1,V2|Vs], Es1, NewEs, SG).
```

◀ TP2 - Assignment Delivery

MT1 (10/11/2023) (Prolog) (English version) ▶